Client-Server Messaging Platform

AQA Computer Science A Level Non-Exam Assessment

Sam Poirier

22 February 2019

Contents

1 Analysis

1.1 Analysis Plan

1.1.1 Interview

It is important to interview the client, Mr Ovayolu; a secondary school teacher, in order to establish the key features that they require for the program to include, as well as what features are less important not needed. I will work from the responses the client gives in order to accurately tailor the software to their requirements.

1.1.2 Questionnaire

I will develop a questionnaire to determine the demand from the general public in order to tailor the software to be desirable to users other than the initial client, as long as the general demand does not contradict the client's needs. I will use google forms to create the questionnaire.

1.1.3 Project Background

Within the Project Background I will detail the specific Client needs, in order to fully flesh out the purpose of the software. I will also analyse a number of existing pieces of software that fulfil similar solutions to determine crucial aspects that should be included in my project, but also determine flaws which I should make sure not to include in the software

1.1.4 Objectives

Under the objectives section, I will detail the exact objectives the project must meet, along with the specific criteria of each objective. This will allow me to ensure that the final product meets with all the client expectations.

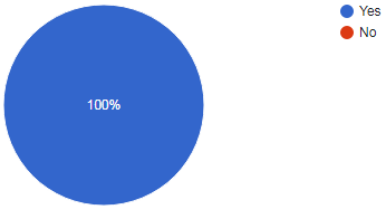
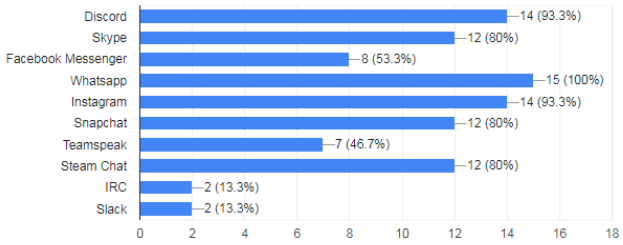
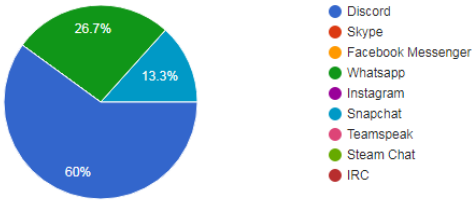
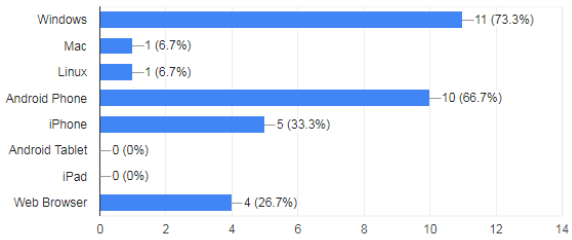
1.1.5 Solutions

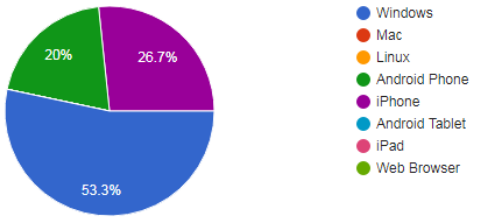
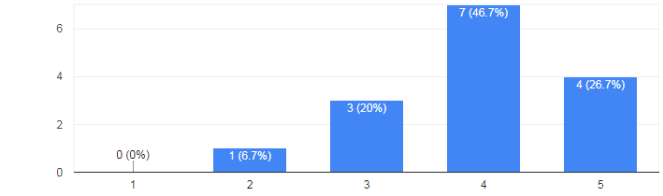
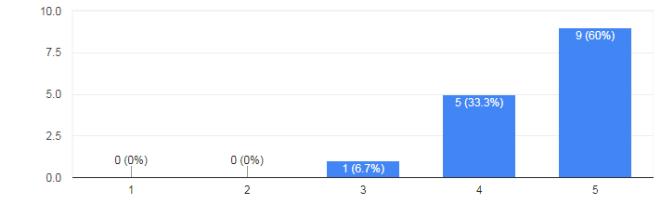
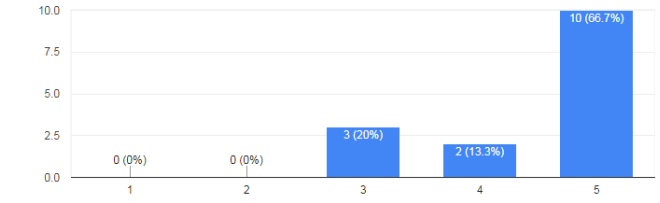
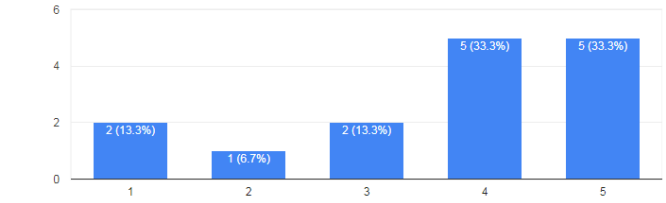
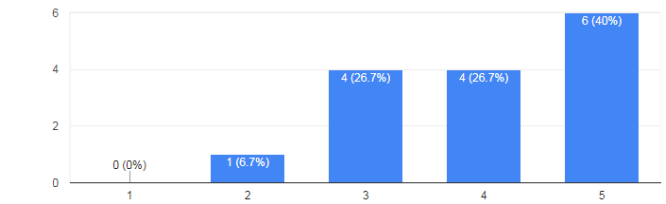
Here I will detail the technical aspects I plan to use in order to develop the software, including programming languages, Integrated Development Environments, general database rules (if applicable) and other technical Solutions.

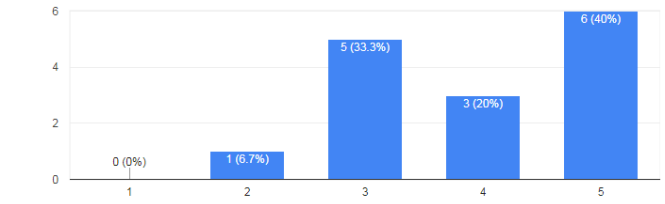
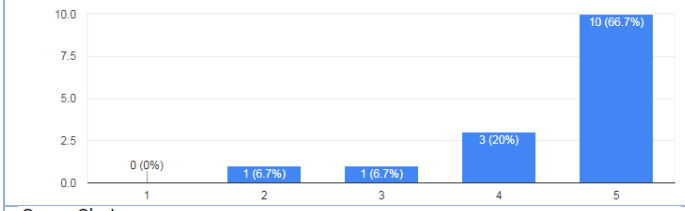
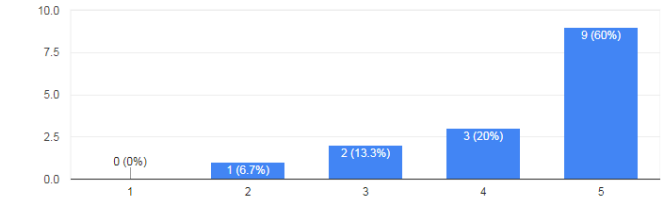
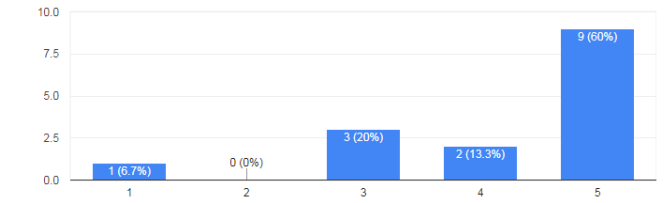
1.2 Interview

Questions	Responses	Analysis
What features of an instant messaging platform are most important for you?	• Fast and secure communication between students by 1:1 or group chat • Easy to understand, clean but functional user interface • Be able to run in all Windows versions • View chat history • Group chat and direct messaging • Reply to a particular message, both in a group chat and in a private chat. • Forward the messages to a particular group chat or an individual. • Get alert when new messages arrived. • View list of connected users in the school. • Report button for inappropriate, bullying, threatening messages…	The software is being designed specifically for this user. It is important that as many of these are implemented as possible, and ideally all of them are.
How often will the users use the platform?	Break and lunch time	The software will mostly be used during the day, so there will be time available for potential server maintenance; there isn't a need for 100% uptime.
How many users should the platform support?	Concurrent around 100, total around 2000	The program must support a large quantity of users – some stress/load testing will be needed to ensure the load can be managed successfully.
How much message history should users be able to see?	3 Months or 100 messages. Ideally, user should be able to set this in settings.	The client wants a configurable number of messages to be readable, meaning the server should likely store all messages then send the client the amount they have requested.

1.3 Questionnaire

Questions/Responses	Analysis
Do you currently use or have you ever used an instant messaging application? 15 responses 	Everyone who answered the survey has used an IM application at some point, which means that the feedback received is not necessarily representative of all the users, so should not be treated as an absolute representation of the userbase.
Please tick/list all Instant Messaging applications that you have used. 15 responses 	WhatsApp is the most widely used IM application, followed closely by Discord and Instagram. The least widely used applications are IRC and Slack. All of these extremes should be investigated in more detail in order to replicate their success and avoid their mistakes.
Which of the previously mentioned applications do you use the most? 15 responses 	Interestingly enough, although WhatsApp was the most widely used, Discord was in fact the one that was used the most overall, and snapchat which was thoroughly in the middle of the widely used, was one of the only three chosen as most used. This shows that the most widely used applications are not necessarily the best ones.
What platforms do you frequently use for Instant Messaging? 15 responses 	Windows and Android Phone are by far the most widely used, and so for the best coverage of users, are the platforms that would be the best to target.

Which of the previously mentioned platforms do you use most? 15 responses 	Again, Windows is by far the most used platform and is therefore the clear frontrunner in terms of OS to develop for.
On a scale of 1-5, how important are these features of an Instant Messaging application to you? Data Security 15 responses 	73.4% of those surveyed indicated that Data Security is important for them. This is clearly an important feature.
Secure Logins 15 responses 	93.3% of users indicated that secure logins where an important feature for them so this also is a crucial aspect of the project.
Viewable Message History 15 responses 	Again, most users agree that having a viewable message history is an important feature of an IM application.
Transparency about what happens to your data/where it is stored 15 responses 	Only just over half of the responses indicated that they cared about transparency regarding their personal data and what was done with it.
Easy to understand User Interface 15 responses 	Most people agreed that an easy to understand/use user interface was a good feature. It could be worth developing a few different layouts to receive feedback about.

User Interface Customisation 15 responses 	A large number of those surveyed indicated that a customisable UI would be wanted. More detail might be needed into what exact customisations are wanted.
Direct Messaging 15 responses 	Most users agreed that Direct Messaging would be an important feature. It was also requested by the Client so is a high priority.
Group Chats 15 responses 	Group chats were also indicated a feature wanted by the client as well as the survey candidates. This means that it should definitely be implemented in the product.
Multimedia Support (image/video embeds, text formatting etc) 15 responses 	A fairly mixed response from the survey, but this was also requested by the client so should be included regardless.
Are any other features are important to you? 6 responses Support for custom commands Voice chat Decent light theme ban hammers/block features, customisation bots, being able to ban Ellhoir A system to recommend friends based on linked accounts and/or other friends (eg Snapchat or Discord) Delete messages at both end...	Some of these feature requests are more in the scope of this project than others. These were individually requested and so are low priority.

1.4 Project Background

Instant Messaging platforms have existed pretty much as long as computers have, originally (in the 1960s) acting as a way of communicating between users on multi-user operating systems. They were also originally used for notification systems like printing. A good example of possibly the first instance of this software is from CTSS (Compatible Time-Sharing System), an operating system developed by MIT which was the first to demonstrate any sort of messaging, and was a precursor to email.

Hashing has also existed as a concept for a long time, with the term originating from its non-technical meaning (to "chop" or "make a mess" out of something), and was first coined in the 1960s. For this project, we will be specifically looking at Cryptographic Hashing which usually consists of a mathematical algorithm that maps data of arbitrary size to a bit string (a hash) and is designed to be a one-way function. The purpose of this function is to obscure a user's personal data so that it is secure.

1.4.1 Specific Client Needs

I will be creating a secure Client-Server messaging platform. My client is Mr Ovayolu, a Computer Science teacher who wishes to set up a secure platform for the teachers and students to discuss work with each other. The focus will be on a secure design, hosted by the client to ensure they know exactly where their messages/data is stored and that it is not being sold to companies by the owners of the software. This is important as a large amount of the data on the system will belong to minors, so the school has extra responsibility to protect them. The end users will be teachers or students who may not be very 'tech-literate'; the application must be easy to set up and use.

1.4.2 Existing Solutions

1.4.2.1 Internet Relay Chat

The first IM platform to run over the internet was Internet Relay Chat (IRC), which was made in 1988. By 1989 IRC had spread across the world with over 40 servers worldwide.

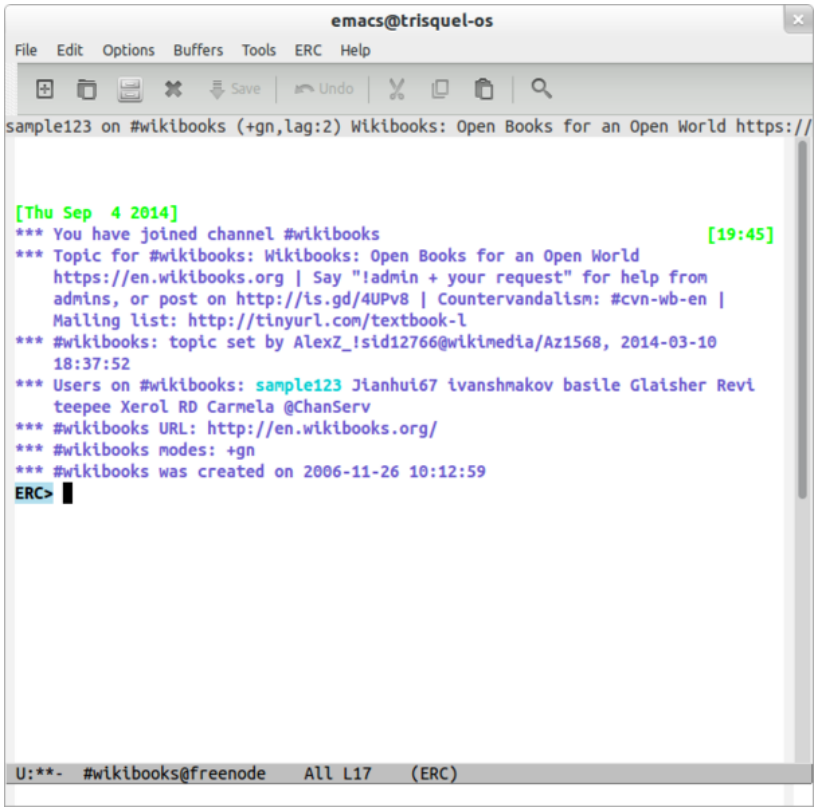


Figure 1.1: An IRC Client called 'ERC' which runs within the text editor Emacs.

Pros	Cons
Client-server model – the client's GUI or functionality can be upgraded without touching the server allowing for all sorts of clients to be compatible with each other.	Complicated to set up servers correctly and securely.
Can be secure if set up correctly.	Some clients are complicated to use and set up.
	Little support for multimedia formats, reporting messages, file transfer etc.

1.4.2.2 Discord

Released in May 2015, Discord is a multiplatform Instant Messaging and voice-over-internet Protocol (VoIP) application. It runs on pretty much every platform available and is free to use.

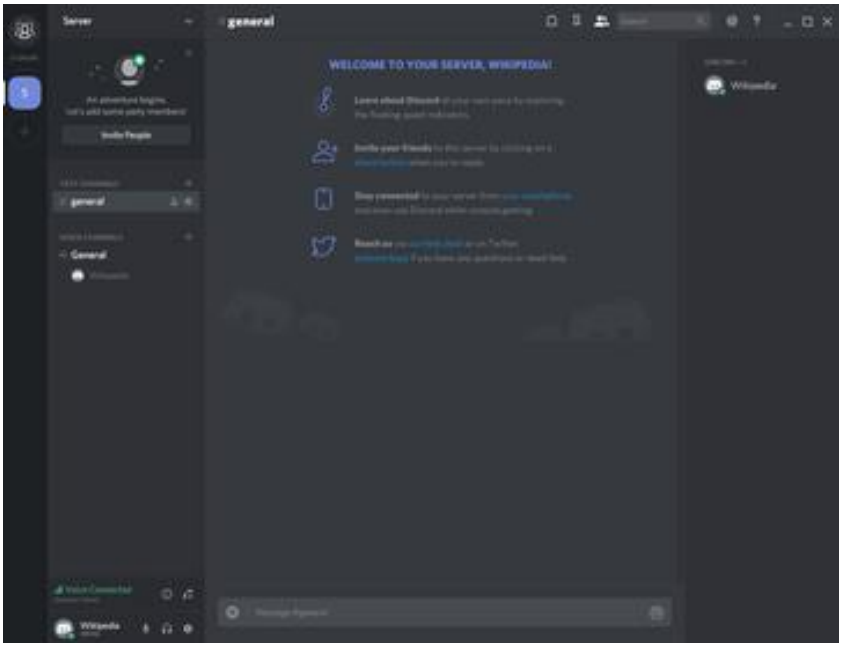


Figure 1.2: The Discord desktop application after creating a new server.

Pros	Cons
Extremely easy to create new accounts and servers – all implementation is abstracted away.	No personally hosted servers – all data is controlled and stored by Discord Inc. meaning that a user is not in control of it's data's security or status.
Servers are secure in the sense that only invited users can join.	Written using Electron – a framework with a famously high memory footprint on the PCs running it, so may not work well on older machines.
Lots of support of extra features, for example VOIP, image embeds, direct messages, file transfer.	

1.4.2.3 WhatsApp

WhatsApp is a free Instant Messaging, Voice and Video Chat client. It was founded in 2009 and is owned by Facebook. It is supported on many platforms including most smart phones.

Figure 1.3: WhatsApp running on an Android Device

Pros	Cons
Easy to use and set up.	Very little privacy or control over data – it is owned by Facebook, a company which is notorious for abuse of its users' personal data.
Easy to connect to new 'servers' or group chats.	User interface is not designed for large groups of people or high rate of messages being sent.
Lots of features including voice chat, file sharing, direct messages etc.	

1.4.2.4 MD5 Hashing Algorithm

MD5 is one in a series of algorithms designed by Professor Ronald Rivest of MIT in 1991. It was designed as a cryptographic hashing function for use in encryption.

Pros	Cons
Implementations exist already in many different programming languages already, in addition to pseudocode for the algorithm being widely available.	Suffers from extensive vulnerabilities, including a high chance of collisions (when two or more different input string produce the same output hash string) rendering it obsolete for encryption usage.

1.4.2.5 SHA-2 Hashing Algorithm

SHA-2 (Secure Hash Algorithm 2) is a set of cryptographic hash functions designed by the United States National Security Agency (NSA) in 2001. It utilises a Merkle–Damgård structure.

Pros	Cons
When implemented correctly is virtually uncrackable.	Low support on older systems, for example Windows XP. This conflicts with the Client's requests of running on all OSes

1.5 System Objectives

No	Objective	Performance Criteria
1	Login Screen User Interface must contain: a) Input for username and password b) Login Button c) Open Register Window Button d) Error Label to display login issues	All of these UI elements must exist in an organised and easy to read layout.
2	Register User Screen UI must contain: a) Input for username, password and password confirmation b) Register (submit) button c) Error label to display register issues	All of these UI elements must exist in an organised and easy to read layout.
3	Main Window UI must contain: a) Box to contain message history b) Input box for sending messages c) Button to send message d) General server information; who's connected, who you're logged in as etc	All of these UI elements must exist in an organised and easy to read layout.

4	Connecting to the server should be secure.	The initial handshake between client and server should be secure; ie no 'fake' clients attempting to connect should be accepted.
5	Client/Server Communication should be secure.	All packets to and from the server should be secure so that anyone intercepting them cannot read the information.
7	User interface should be easy to use and read.	The UI should be intuitive to read, use and understand.
8	Client and server should be easy to set up and run.	The client and server should require minimal setup to get into full working/running order.
9	The server must be able to handle lots of users both consecutively and in general.	The server must be able to support multiple concurrently connected users and a larger amount of total registered users.
10	There must be a way for users to configure settings for the client.	The clients should be configurable via a settings menu which is saved between sessions.
11	The application must support most requested features by the client.	As many features requested by the client as possible should be implemented into the program.
12	Users should be able to edit messages	All messages sent by the user should be editable by them, but users should not be able to edit other's messages .
13	Users should be able to report messages	Users should be able to report messages in case they contain rude/innapropriate words etc.
14	Admins should be able to see user reports	The admin should have an administration menu which they can use to review the reports against users.
15	Admins should be able to see user statistics	Admin accounts should be able to see basic statistics for each user account.
16	Admins should be able to promote/demote users	Admin accounts should be able to give admin permissions to other accounts and remove admin permissions from other accounts.
17	Admins should be able to delete all messages	Admin accounts should be able to delete all messages as they could contain rude/explicit content.
18	Logged in users list should be displayed alphabetically	The list of online users should always be automatically sorted to display connected users in alphabetic order rather than join order.
19	Users should be able to style their messages	Users should be able to format their messages to feature bold, italics etc using a markdown like format (surrounding it with symbols, eg _italics_ would display as *italics*).

20	Hovering over messages should highlight them	Hovering over a message should highlight the background making it easier to read and determine which one you are interacting with.

1.6 Solutions

1.6.1 Framework

Language	Pros	Cons
Python	Lightweight Simple Syntax Can be developed on school computers Multiplatform Decent documentation	Interpreted therefore slow and not memory efficient
C#	Powerful Personally preferred language Lots of good documentation Very easy to debug	Low multiplatform support Cannot be developed on school machines
JavaScript (via Electron)	Lightweight Forgiving syntax Multiplatform	Extremely loosely typed – easy to make bugs Hard to debug

1.6.2 GUI Framework

Language	Framework	Pros	Cons
Python	tKinter	Shipped stock with python, no extra installation	Extremely limited in terms of functionality Hard to use
	PyQt	Support for advanced GUI configuration Easy to use	C++ Backend, some errors are hard to catch and result in silent crashes
C#	Winforms	Easy to use Stock with C#	Lots of auto-generated code or lots of manual setup
JavaScript	Electron (with node.js, jQuery and npm)	Frontend developed in HTML/CSS so the GUI is very customisable.	Very memory inefficient Uses lots of other people's modules/code

1.6.3 Integrated Development Environments

IDE	Pros	Cons
Visual Studio	Advanced Debugging tools Good at identifying programming errors before compiling/running Easy to use	'Heavy' piece of software – lots of overhead
Visual Studio Code	Advanced Debugging tools Good at identifying programming errors before compiling/running Easy to use Lightweight	Less features than Visual Studio
IDLE	Comes stock with most python installations Very lightweight Simple Debugging tools	Feature lacking; little auto completion or pre-run bug finding
Notepad++	Extremely lightweight	Very few features Cannot compile many languages

1.6.4 Database Formats

Format	Pros	Cons
SQL	Very powerful Lots of supported datatypes Can create databases with complex structures	Complicated to set up 'Heavy' – not very lightweight at all
SQLite	Lightweight Reasonable amount of datatypes Support for reasonably complicated structures	Less features than SQLite
MongoDB	Unique Solution; structure is based upon a tree rather than a table	Feature lacking; little auto completion or pre-run bug finding

1.6.5 Solution Decisions

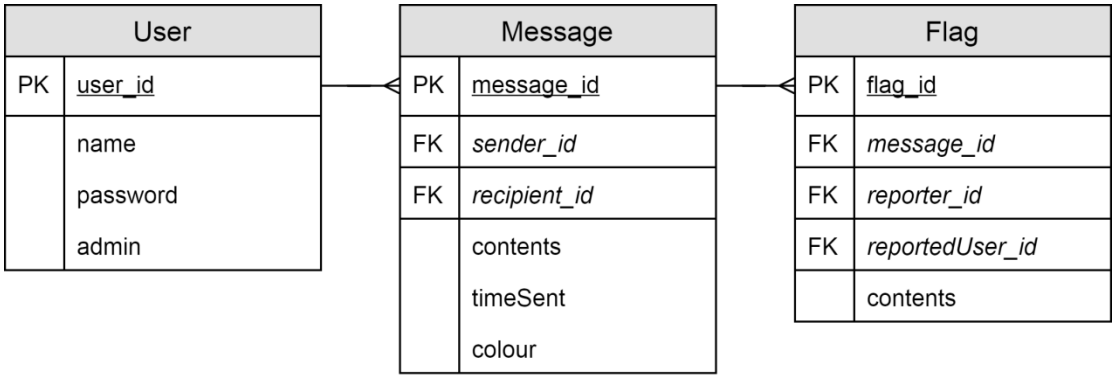
I plan to write my program in Python, as it runs on multiple platforms easily and can be debugged without too much trouble. I will use IDLE to develop the project as it has all the features I need, and is nice and lightweight without too much bloat. I will use PyQt for my GUI as tKinter is not advanced or powerful enough to meet the client needs. I will use SQLite

for my database as it will likely be quite simple; SQLite is a good compromise between features and functionality.

2 Documented Design

2.1 Database Structure & SQL Statements

2.1.1 Entity Relationships and Attributes



2.1.2 Normalised Data Structures

User(user_id, name, password, admin)
Message(message_id, *sender_id*, *recipient_id*, contents, timeSent, colour)
Flag(flag_id, *message_id*, *reporter_id*, *reportedUser_id*, contents)

2.1.3 Data Dictionaries

Table Name	User			
Primary Key	user_id			
Foreign Keys	N/A			
Data Item	**Data Type**	**Validation**	**Sample Data**	**Description**
name	String	Not null	“user”	The username.
password	String	Not null	“O.L&ZXJL3=!VK&\> @,!”	The (one way) hashed password.
admin	Boolean	Not null	False	Indicates whether this account has admin privileges.

Table Name	Message			
Primary Key	message_id			
Foreign Keys	sender_id, recipient_id			
Data Item	**Data Type**	**Validation**	**Sample Data**	**Description**
sender_id	Integer	Not null	2	The id of the user who sent the message.

recipient_id	Integer	Not null	1	The id of the recipient of the message (1 is the server which means everyone).
contents	String	Not null	“Hello World”	The content of the message.
timeSent	String	Not null	“26-10-00 11:10”	The date/time the message was sent.
colour	String	Not null	“#000000”	The hex colour value to display the message as.

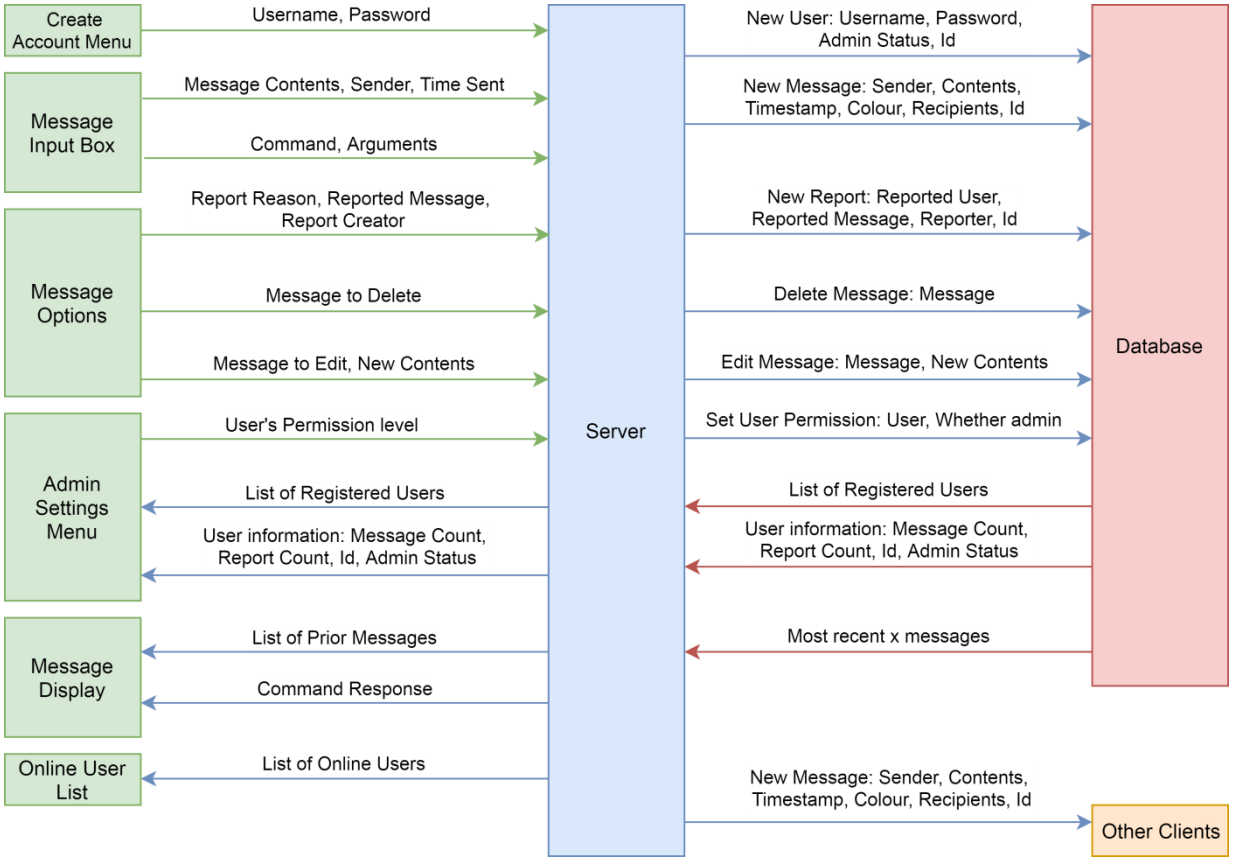
Table Name	Flag			
Primary Key	user_id			
Foreign Keys	message_id, reporter_id, reportedUser_id			
Data Item	**Data Type**	**Validation**	**Sample Data**	**Description**
message_id	Integer	Not null	1	The id of the message that was reported.
reporter_id	Integer	Not null	2	The id of the user who reported the message.
reportedUser_id	Integer	Not null	3	The user who’s message was reported.
conents	String	Not null	“They insulted me”	The reason why the message was reported .

2.1.4 SQL Statements

Statement	Use
`SELECT * FROM User`	Extracts all users in order to verify logins.
`SELECT * FROM Message limit ? offset (SELECT count(*) FROM Message)-?`	Selects x most recent message from the database where ? is the amount to select.
`SELECT name FROM User WHERE user_id == ?`	Gets the username of a user from their id.
`SELECT name FROM User WHERE name == ?`	Used to check if a user with a certain name exists, returns null if they do not.
`SELECT user_id, name, admin FROM User WHERE name != 'SERVER'`	Gets a list of all registered users apart from the ‘SERVER’ user as that is not a true user.
`SELECT user_id, admin FROM User WHERE name == ?`	Gets a user’s user id and whether they’re an admin from their name.
`SELECT count(*) FROM Message WHERE sender_id == ?`	Returns the amount of messages a specific user has sent.
`SELECT * FROM Flag WHERE reportedUser_id == ?`	Selects all the reports corresponding to a specific user.
`SELECT contents FROM Message WHERE message_id == ?`	Gets the contents of a message from its id.
`INSERT into User(name, password) values (?, ?)`	Creates a new non admin user, as ‘admin’ has a default value of False.
`INSERT into Message(sender_id, contents, timeSent, recipient_id, colour) values (?,?,?,?,?)`	Creates a new message.
`SELECT last_insert_rowid()`	Gets the Id (primary key) of the most recent insert.
`SELECT sender_id FROM Message where message_id == ?`	Gets the sender id of a message from the message Id.

`INSERT into Flag(reportedUser_id, message_id, reporter_id, reportReason) values (?,?,?,?)`	Creates a new message report.
`UPDATE Message SET contents=? WHERE message_id=?`	Changes a message's contents by Id.
`UPDATE Message SET contents=?,sender_id=?,colour=? WHERE message_id=?`	Changes a message's contents, who sent it and its colour. Used to 'delete' messages.
`UPDATE User SET admin=? WHERE user_id=?`	Sets a users' privelidges.

2.1.5 Data Flow Diagram

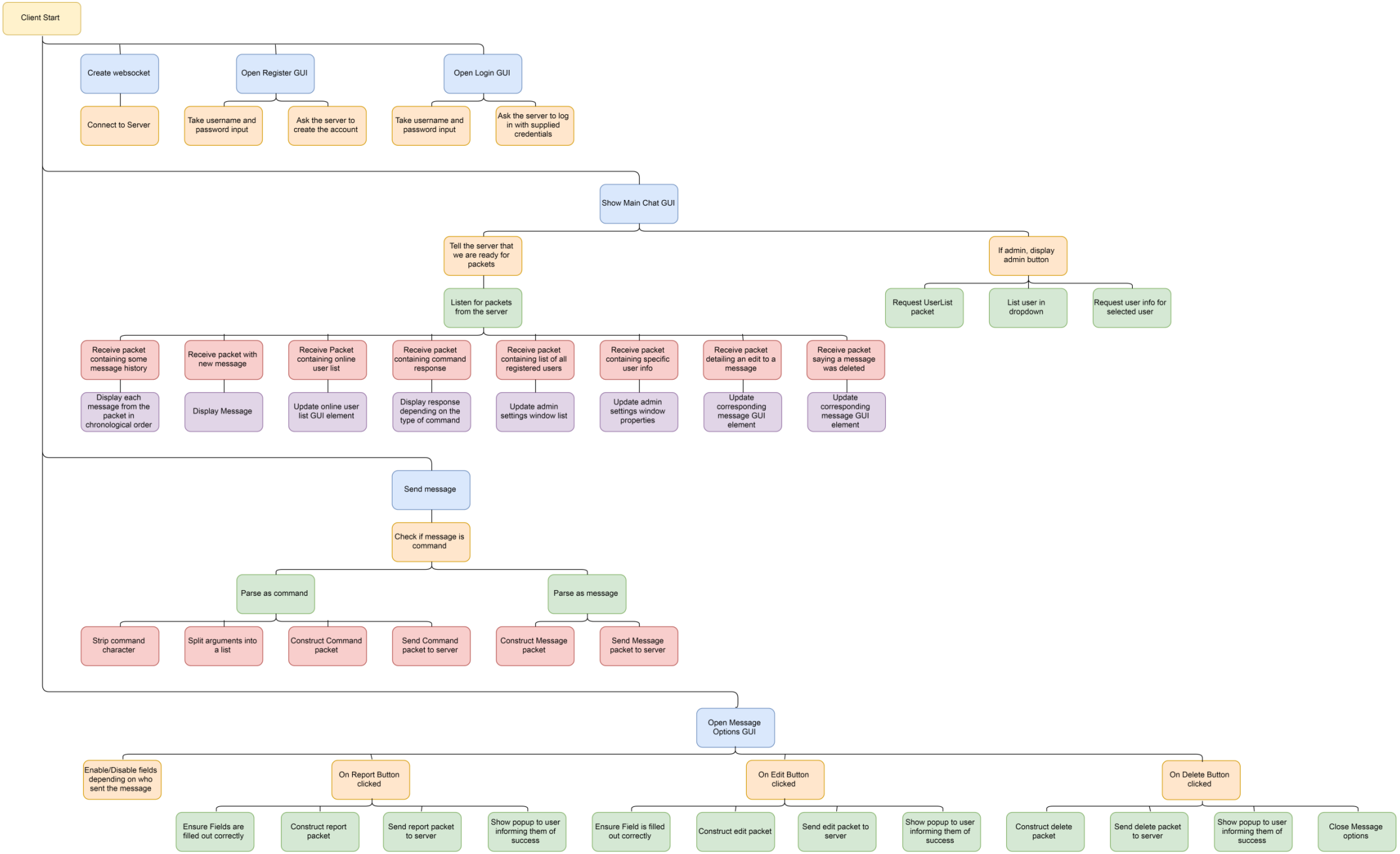


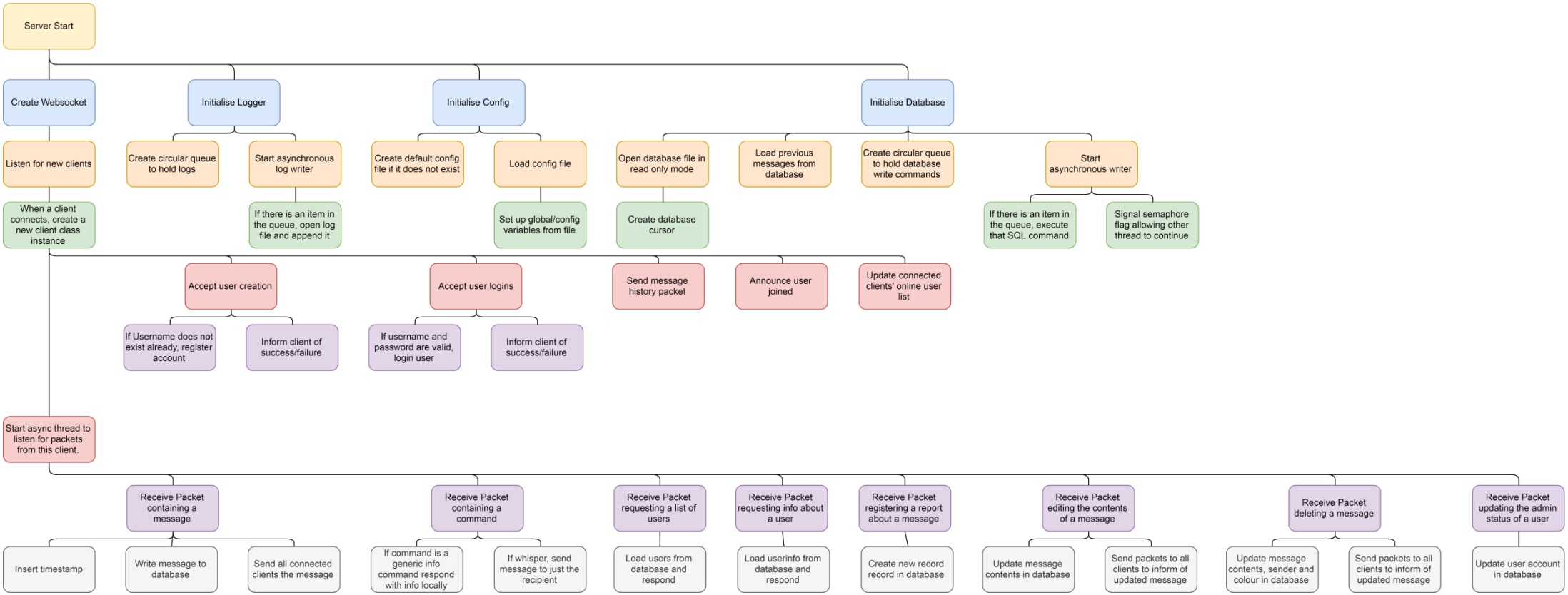
2.2 System Design

2.2.1 IPSO Chart

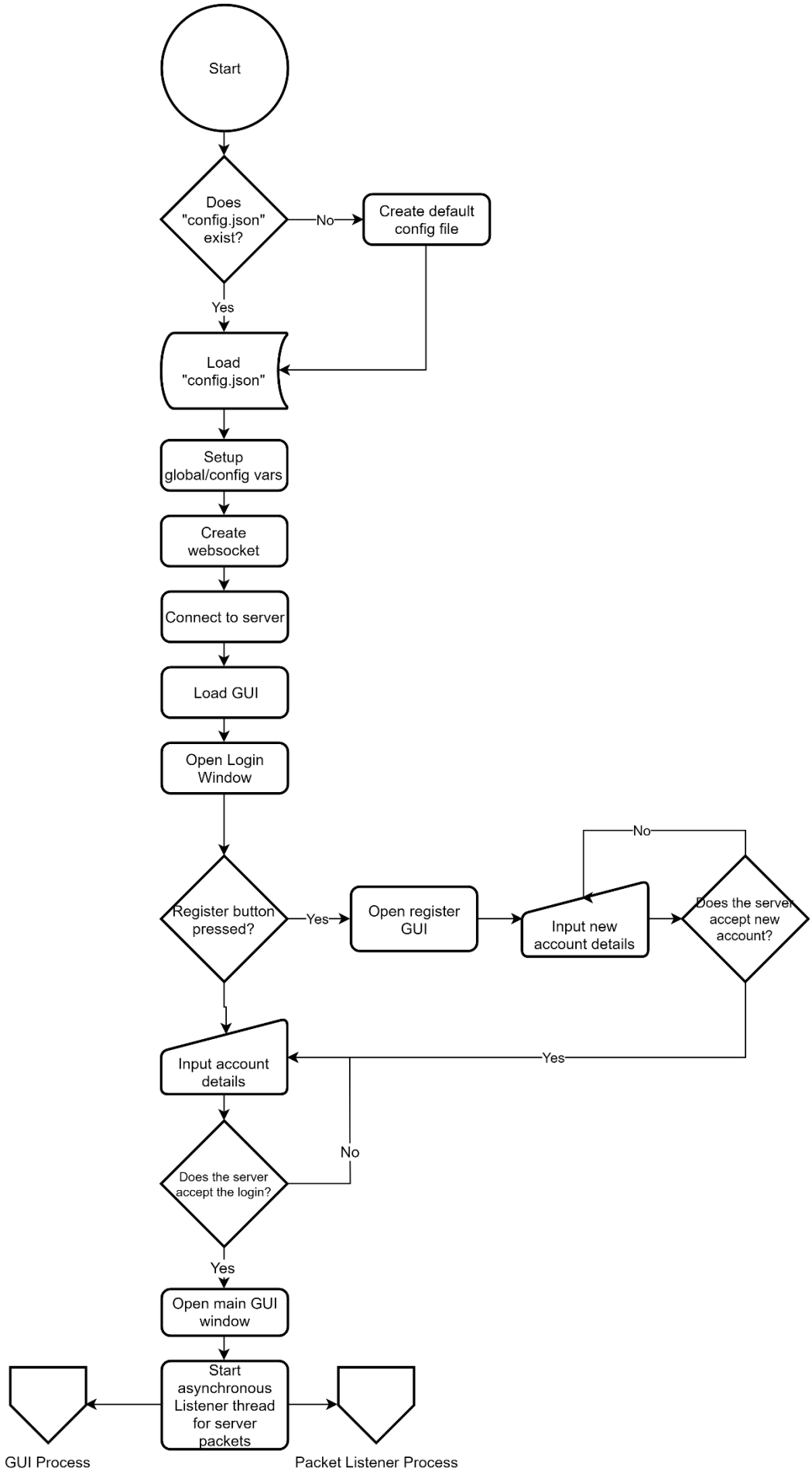
IPSO	Program Section	Item
Input	Registering User	Username Password Confirm Password
	Logging in	Username Password
	Sending Message	Message Contents
	Reporting User	Report Reason
Processing	Registering User	Check if username is available Check passwords match
	Logging in User	Check username and passwords match
	Sending Message	Check if command Insert username Insert timestamp Attach colour Designate recipients Send to other clients
	Process command	Determine if valid command Generate corresponding response Return response to correct recipients
	Reporting User	Attach message id Attach sender id
Storage	Register User	Create new record in User table in database
	Sending Message	Create new record In Message table in database
	Reporting User	Create new record in Flag table in database
	Promoting user to admin	Update user record in User database table
	Editing message	Edit message record in Message database table
	Deleting message	Edit message record in Message database table
Output	Display Message	Create new message widget Update text elements
	Display Online Users	Update online users widget

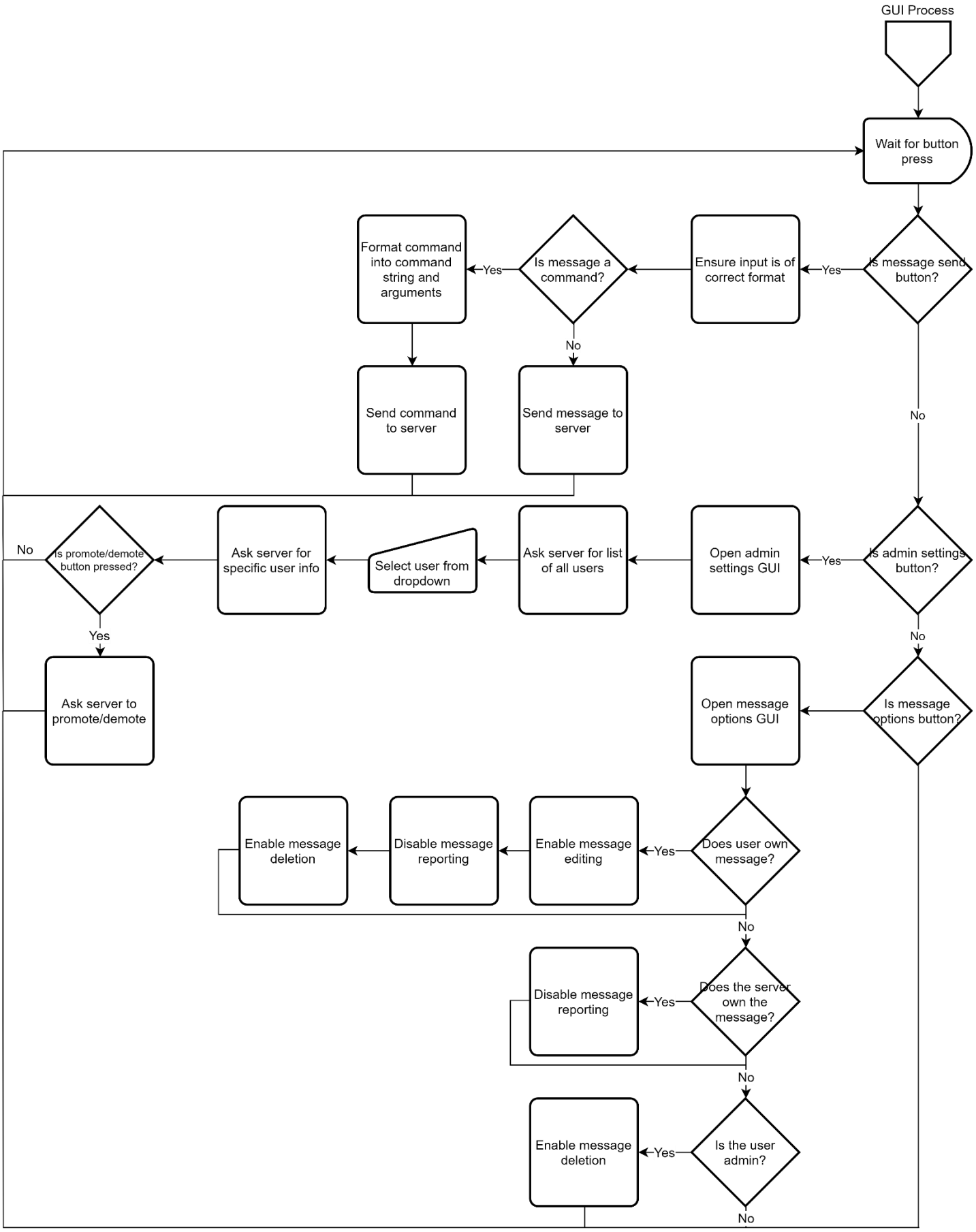
2.2.2 Top-down diagrams

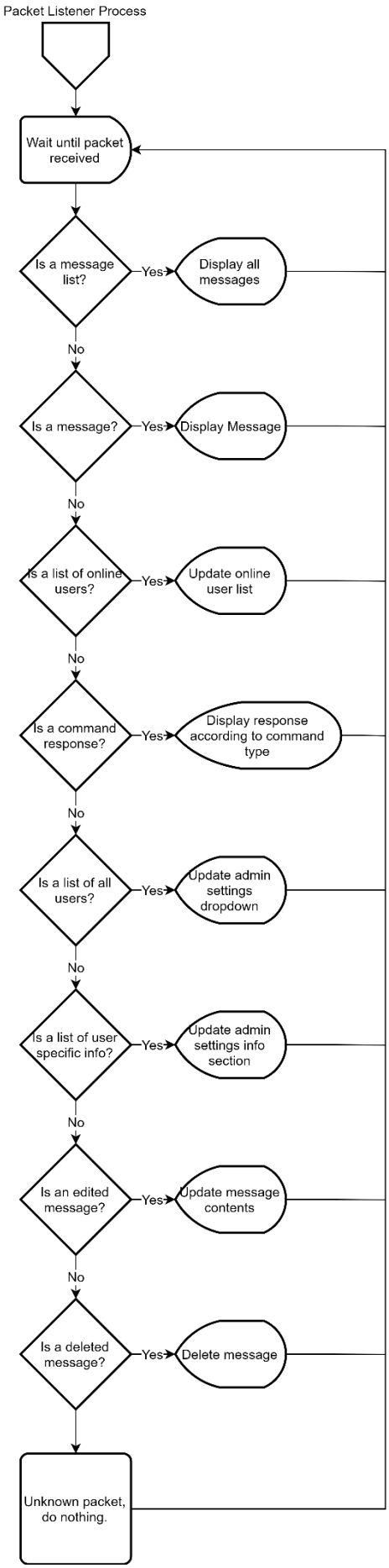



Server Start
Create Websocket
Initialise Logger
Initialise Config
Initialise Database
Listen for new clients
Create circular queue to hold logs
Start asynchronous log writer
Create default config file if it does not exist
Load config file
Open database file in read only mode
Load previous messages from database
Create circular queue to hold database write commands
Start asynchronous writer
When a client connects, create a new client class instance
If there is an item in the queue, open log file and append it
Set up global/config variables from file
Create database cursor
If there is an item in the queue, execute that SQL command
Signal semaphore flag allowing other thread to continue
Accept user creation
Accept user logins
Send message history packet
Announce user joined
Update connected clients' online user list
If Username does not exist already, register account
Inform client of success/failure
If username and password are valid, login user
Inform client of success/failure
Start async thread to listen for packets from this client.
Receive Packet containing a message
Receive Packet containing a command
Receive Packet requesting a list of users
Receive Packet requesting info about a user
Receive Packet registering a report about a message
Receive Packet editing the contents of a message
Receive Packet deleting a message
Receive Packet updating the admin status of a user
Insert timestamp
Write message to database
Send all connected clients the message
If command is a generic info command respond with info locally
If whisper, send message to just the recipient
Load users from database and respond
Load userinfo from database and respond
Create new record record in database
Update message contents in database
Send packets to all clients to inform of updated message
Update message contents, sender and colour in database
Send packets to all clients to inform of updated message
Update user account in database

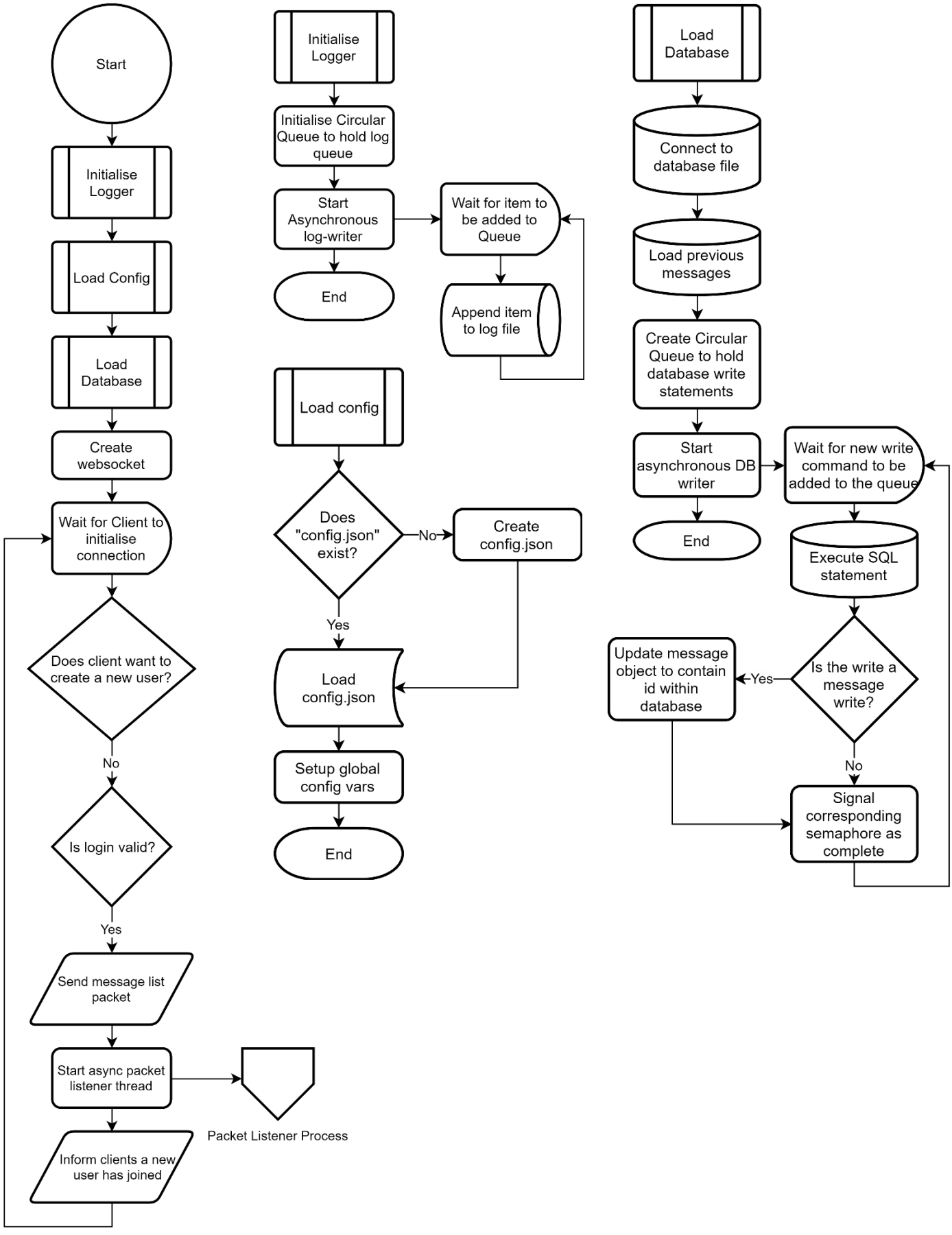
2.2.3 Client Flowchart

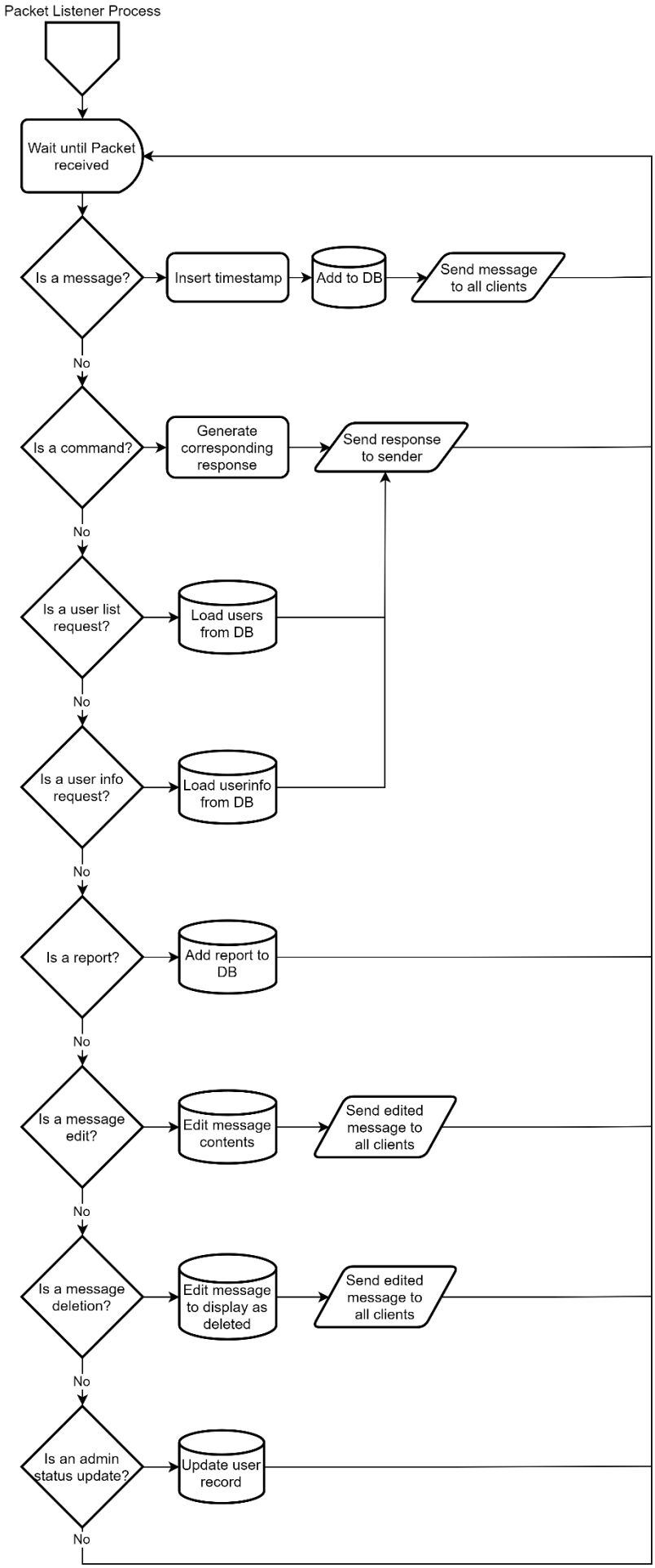



GUI Process
Wait for button press
Is message send button?
Yes
Ensure input is of correct format
Is message a command?
Yes
Format command into command string and arguments
No
Send command to server
Send message to server
No
Is admin settings button?
Yes
Open admin settings GUI
Ask server for list of all users
Select user from dropdown
Ask server for specific user info
Is promote/demote button pressed?
No
Yes
Ask server to promote/demote
No
Is message options button?
Open message options GUI
Does user own message?
Yes
Enable message editing
Disable message reporting
Enable message deletion
No
Does the server own the message?
Yes
Disable message reporting
No
Is the user admin?
Yes
Enable message deletion
No


Packet Listener Process
Wait until packet received
Is a message list?
Yes
Display all messages
No
Is a message?
Yes
Display Message
No
Is a list of online users?
Yes
Update online user list
No
Is a command response?
Yes
Display response according to command type
No
Is a list of all users?
Yes
Update admin settings dropdown
No
Is a list of user specific info?
Yes
Update admin settings info section
No
Is an edited message?
Yes
Update message contents
No
Is a deleted message?
Yes
Delete message
Unknown packet, do nothing.

2.2.4 Server Flowchart



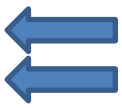

Packet Listener Process
Wait until Packet received
Is a message?
Insert timestamp
Add to DB
Send message to all clients
No
Is a command?
Generate corresponding response
Send response to sender
No
Is a user list request?
Load users from DB
No
Is a user info request?
Load userinfo from DB
No
Is a report?
Add report to DB
No
Is a message edit?
Edit message contents
Send edited message to all clients
No
Is a message deletion?
Edit message to display as deleted
Send edited message to all clients
No
Is an admin status update?
Update user record
No

2.3 Data Structures & Dictionary

2.3.1 Circular Queue

Both the database writer and logger will operate using a circular queue to ensure there is never an error due to two threads attempting to write at once. The queue will hold each write to be made chronologically, with the oldest item at the front.

Index	Data Example
0	`INSERT into Message ...`
1	`Update User ...`
2	
3	
⋮	
n	



Variables will be needed for the FrontPointer, RearPointer and Length in order to keep track of the size. Custom methods to EnQueue and DeQueue will be written rather than accessing the data array directly.

2.3.2 Client Data Dictionary

Data Item	Data Type	Validation	Sample Data/Info
_app	QApplication		The instance of the PyQy app
_mainGui	MainWindow		The main GUI window
_serverSocket	socket		The websocket that all server connections will go through
_username	String	Not Null	“admin”
_userId	Integer	Not Null	1
_admin	Boolean	Not Null	True
NONPRINTINGCHAR	Char (1 length string)	==“\u200B”	“\u200B”
MAXTRANSMISSIONSIZE	Integer	Not Null	40960
COMMANDCHAR	Char (1 length string)	Not Null	“/”
DEBUG	Boolean	Not Null	False
New message	String	Not Null	“Hello World”
New Username	String	Not Null	“Sam”
New Password	String	Not Null	“secret”
Confirm Password	String	== New Password	“secret”
Username Login	String	Not Null	“User”
Password Login	String	Not Null	“shhhhh”
Report User	String	Not Null	“Said a rude word”
Edit Message	String	Not Null	“Hello World!”

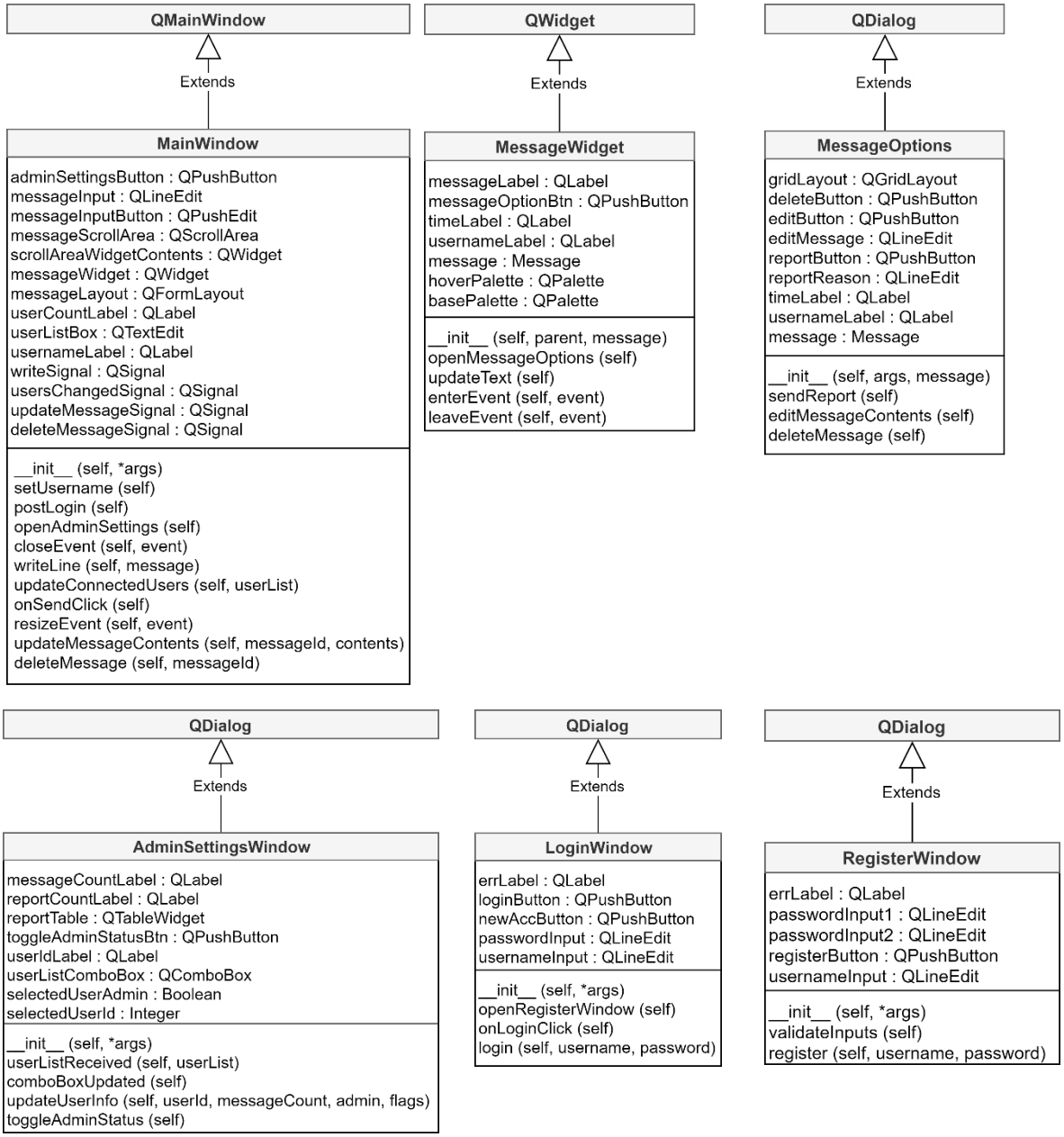
2.3.3 Server Data Dictionary

Data Item	Data Type	Validation	Sample Data/Info
_clients	List of Client	Not Null	A list containing all connected clients
_database	Database	Not Null	Database management class

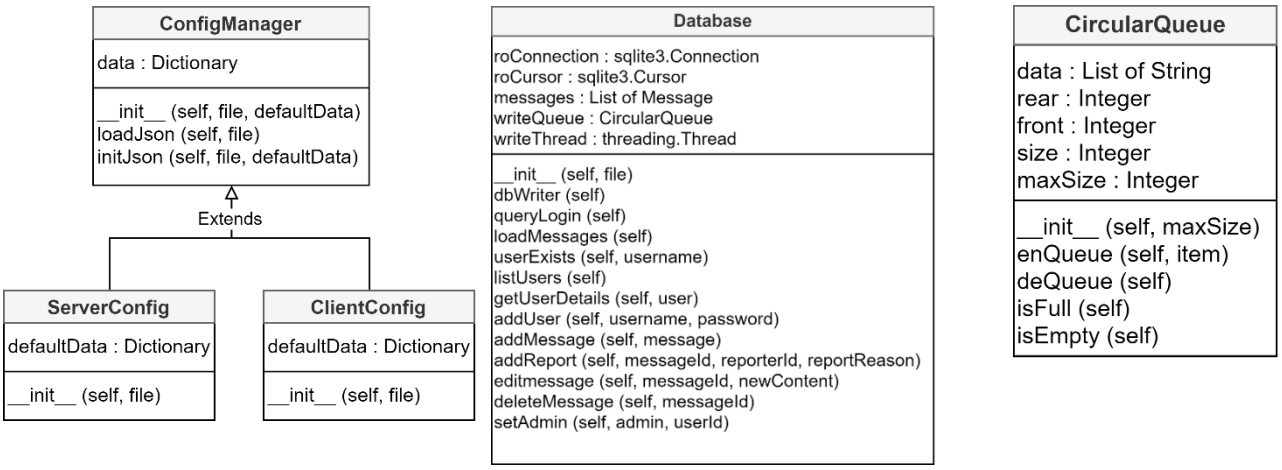
_logger	Logger	Not Null	Logger Class
_configManager	ConfigManager	Not Null	Config Manager Class
INFOLOGGINGENABLED	Boolean	Not Null	False
MAXTRANSMISSIONSIZE	Integer	Not Null	40960

2.4 Class Design

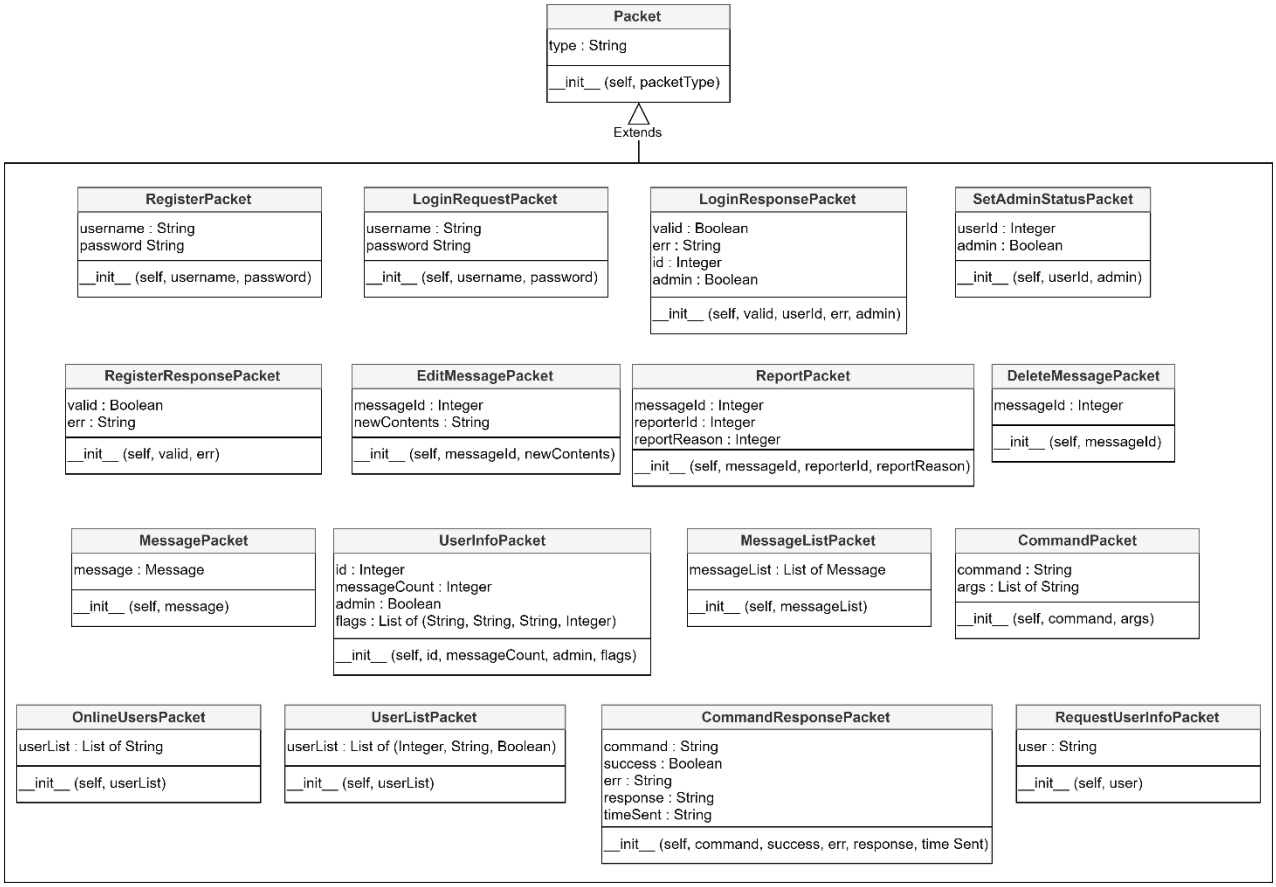
2.4.1 Client Classes

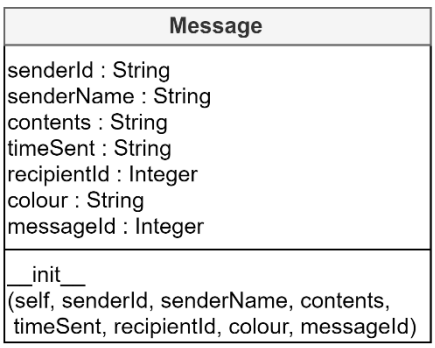


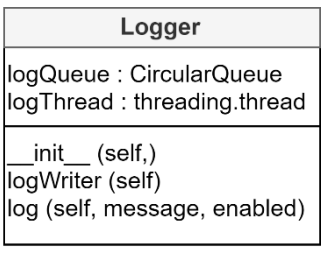
2.4.2 Server Classes



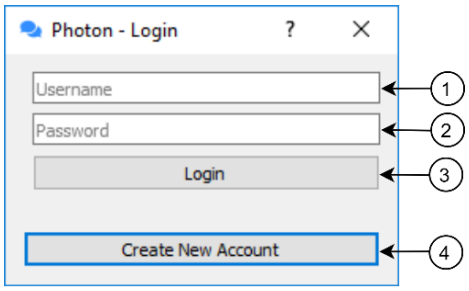
2.4.3 Generic Classes



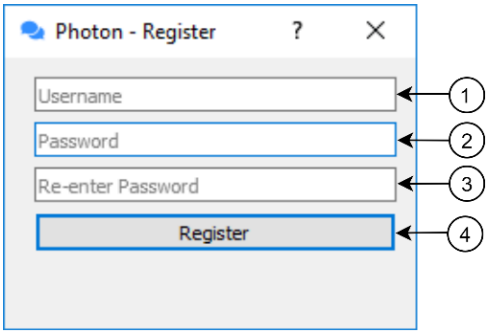




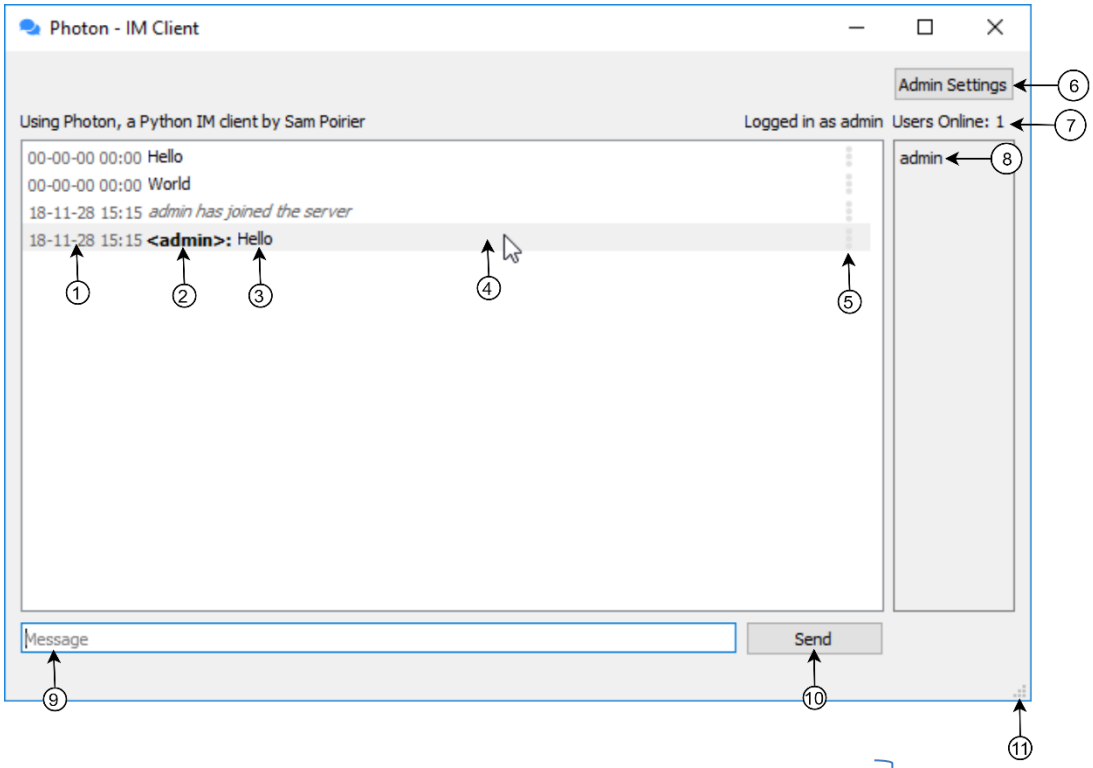
2.5 User Interface Design



1. Input for the username of the account to login to.
2. Input for the password of the account. Displays text as '•' characters.
3. Login button sends login request to server, taking the input from the text boxes.
4. Create new account button opens up the register window, allowing users to create a new account.



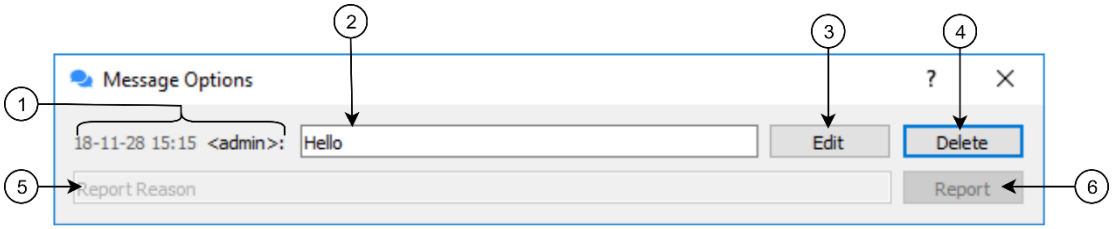
1. Input for the username of the new account.
2. Input for the password of the new account. Displays text as '•' characters.
3. Input for the confirm password of the new account. Displays text as '•' characters. Must equal the box above.
4. Submits the account details to the server, attempting to create a new account



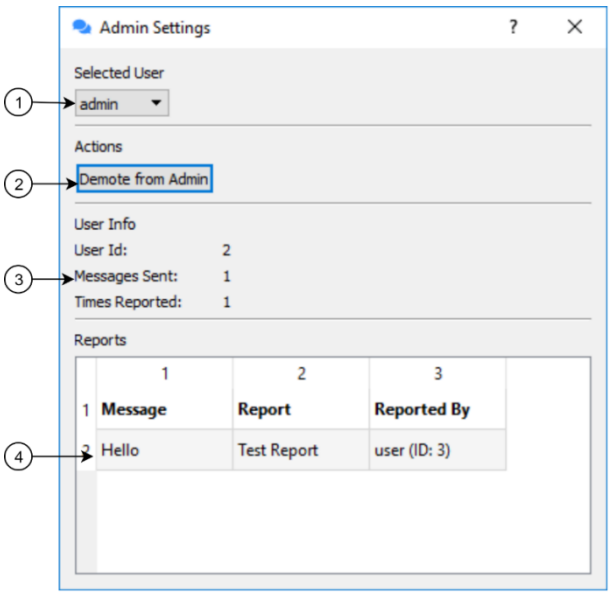
1. Label to display message timestamp.
2. Label to display message sender username.
3. Label to display actual message contents.
4. When hovered, the message widget darkens for ease of reading.
5. Button to open message options widget.

Part of the message widget

6. Button to open admin settings window. Only visible if logged in as admin.
7. Label to display count of online users.
8. Multiline label to display list of online users.
9. Text input to write new messages in.
10. Button to send messages. Also empties Message text input.
11. Window is fully resizable, with all elements scaling properly.



1. Timestamp & Sender display labels.
2. Text input to edit messages. Only enabled if user sent the message.
3. Submits message edit. Only enabled if user sent the message.
4. Deletes message. Only enabled if admin or sender of message.
5. Text input to register a report. Disabled on own or server messages.
6. Button to submit report.



1. Dropdown contains all registered users. Used to select user to edit.
2. Promotes/Demotes a user to/from admin.
3. Labels to display statistics/information about the selected user.
4. If a user has been reported, reports are loaded into this table for admins to look over.

2.6 Algorithms

2.6.1 Circular Queue

The Circular Queue Class is used for the database write queue and the logger in the server file.

```
class CircularQueue():
  """
  A custom circular queue implementation.
  """
  def __init__(self, maxSize):
    if maxSize < 1:
      raise ValueError("Queue size must be at least 1")
    self.data = [''] * maxSize
    self.rear = -1
    self.front= 0
    self.size = 0
    self.maxSize = maxSize

  def enQueue(self, item):
    """
    Add an item to the queue.

    Args:
      item (*): Item to add to the queue.
    """
    if self.size == self.maxSize:
      raise ValueError("Cannot enqueue when the queue is full")
    else:
      self.rear = (self.rear + 1) % self.maxSize
      self.data[self.rear] = item
      self.size = self.size + 1

  def deQueue(self):
    """
    Removes an item from the queue.

    Returns:
      (*): The item at the front of the queue.
    """
    self.front += 1
    self.size -= 1
    return self.data[self.front-1]

  def isFull(self):
    """ Determines if the queue is full. """
    return self.size == self.rear

  def isEmpty(self):
    """ Determines if the queue is empty. """
    return self.size == 0
```

2.6.2 Hashing Algorithm

The hashing algorithm is one-directional and is used for encrypting passwords before storing them in the database.

```
1.  def hashString(string):
2.    """
3.    Custom implementation of a simple one way hashing algorithm
4.
5.    Args:
6.      string (string): The string to hash.
7.
8.    Returns:
9.      (string): The hashed string.
10.   """
11.   bitValueChunk = ""
12.   bitSum = 0
13.
14.   for char in string:
15.     bitSum += ord(char)
16.     bitValue = format(ord(char), 'b') # Convert char to binary
17.     bitValueChunk += bitValue # Append to 'binary chunk'
18.
19.   n = 9
20.   bitValues = [bitValueChunk[i:i+n] for i in range(0, len(bitValueChunk), n)
    ] # Split 'binary chunk' into list of 9 bit binary numbers
21.
22.   moddedBitChunk = ""
23.   for bitValue in bitValues:
24.     bitValue = int(bitValue)
25.     moddedBitValue = bitValue + (bitValue % 37) # Modulo is a one way functi
    on, so we modulo by a prime as the core of the hash. This is the step that e
    nsures the hash is unidirectional
26.     moddedBitValue = moddedBitValue * bitSum # Multiply by sum of the ascii
    values of the chars to ensure similar input strings look different
27.     moddedBitChunk += format(moddedBitValue, 'b') # Convert into binary agai
    n
28.
29.   n = 6 # 6 bit chunks to avoid strange characters
30.   moddedBitValues = [moddedBitChunk[i:i+n] for i in range(0, len(moddedBitCh
    unk), n)] # Split modded 'binary chunk' into list of 8 bit binary numbers
31.
32.   hashed = ""
33.   for moddedBitValue in moddedBitValues:
34.     hashed += chr(int(moddedBitValue, 2) + 33) # Convert binary value into d
    ecimal value then into the corresponding character, skipping the first 33 as
     they are non-printing/whitespace
35.
36.   return hashed
```

2.6.3 Integer Merge Sort

The integer merge sort is not actually used anywhere in the program, but was written as a prototype in order to get the string merge sort working, as they are very similar algorithms.

```
1.  def integerMergeSort(mergelist):
2.    """
3.    Custom implementation of a mergesort algorithm for integers.
4.
5.    Args:
6.      mergeList (list of int): The list to sort.
7.
8.    Returns:
9.      (list of int): The sorted list.
10.   """
11.   if len(mergelist) > 1:
12.       mid = len(mergelist) // 2 # Perform integer division
13.       lefthalf = mergelist[:mid] # Left half of merglist into lefthalf
14.       righthalf = mergelist[mid:] # Right half of merglist into righthalf
15.       lefthalf = MergeSort(lefthalf)
16.       righthalf = MergeSort(righthalf)
17.
18.       i = 0
19.       j = 0
20.       k = 0
21.       while i < len(lefthalf) and j < len(righthalf):
22.           if lefthalf[i] < righthalf[j]:
23.               mergelist[k] = lefthalf[i]
24.               i += 1
25.           else:
26.               mergelist[k] = righthalf[j]
27.               j += 1
28.           k += 1
29.
30.       # Check if left half has elements not merged
31.       while i < len(lefthalf):
32.           mergelist[k] = lefthalf[i] # If so, add to mergelist
33.           i += 1
34.           k += 1
35.       # Check if right half has elements not merged
36.       while j < len(righthalf):
37.           mergelist[k] = righthalf[j] # If so, add to mergelist
38.           j += 1
39.           k += 1
40.   return mergelist
```

2.6.4 String Merge Sort

The String Merge Sort is used to sort the list of online users alphabetically.

```
def stringListMergeSort(mergelist):
  """
  Custom implementation of a mergesort algorithm for strings.

  Args:
    mergeList (list of string): The list to sort.

  Returns:
    (list of string): The sorted list.
  """
  if len(mergelist) > 1:
    mid = len(mergelist) // 2 # Perform integer division
    lefthalf = mergelist[:mid] # Left half of merglist into lefthalf
    righthalf = mergelist[mid:] # Right half of merglist into righthalf
    lefthalf = stringListMergeSort(lefthalf)
    righthalf = stringListMergeSort(righthalf)

    i = 0
    j = 0
    k = 0
    while i < len(lefthalf) and j < len(righthalf):
        l = 0
        while ord(lefthalf[i][l]) == ord(righthalf[j][l]) and l < len(leftha
lf[i])-1 and l < len(righthalf[j])-
1: # If the charachers are the same, we must look at the next one until the
end
            l += 1
        if ord(lefthalf[i][l]) < ord(righthalf[j][l]):
            mergelist[k] = lefthalf[i]
            i += 1
        else:
            mergelist[k] = righthalf[j]
            j += 1
        k += 1

    # Check if left half has elements not merged
    while i < len(lefthalf):
        mergelist[k] = lefthalf[i] # If so, add to mergelist
        i += 1
        k += 1
    # Check if right half has elements not merged
    while j < len(righthalf):
        mergelist[k] = righthalf[j] # If so, add to mergelist
        j += 1
        k += 1
  return mergelist
```

3 Technical Solution

3.1 Server

3.1.1 main.py

```
1.  # Main Server file
2.
3.  import socket
4.  from threading import *
5.  import select
6.  import re
7.
8.  # Load classes and functions from shared libs
9.  import sys
10. sys.path.insert(0, '../Libs')
11. from packets import *
12. from photonUtilities import *
13. from database import *
14. from logger import *
15. from configManager import *
16.
17.
18.
19. # Global Variables
20.
21. _clients  = []
22. _database = None
23. _logger = None
24. _configManager = None
25.
26. INFOLOGGINGENABLED = None
27. MAXTRANSMISSIONSIZE = None
28.
29.
30.
31. # Classes
32.
33. class Client:
34.   """
35.   Represents one client that is connected to the server. A new instance is m
    ade for each connection, each client only interacts with their corresponding
     thread.
36.
37.   Properties:
38.     socket (socket.socket): The websocket that this client is connected thro
    ugh.
39.     address (string): The IP of the connected client.
40.     id (int): the id of the connected client (not constant between sessions)
    .
41.     thread (threading.thread): The asyncronous listener thread for this clie
    nt.
42.     username (string): The username of the user.
43.     userid (int): The id of the user account (constant between sessions).
44.     admin (bool): Denotes whether the user account is admin.
45.   """
46.   def __init__(self, clientSocket, clientAddress):
47.     """
48.     Initialises the connection process, starts asynchronous listener thread
    and handles login handshake and account creation
49.
50.     Args:
51.       clientSocket (socket.socket): the websocket that the connection is han
    dled over.
```

```
52.       clientAddress (string, int): The IP and id of the connected client.
53.
54.     ToDo:
55.       Break this up!
56.     """
57.     try:
58.       global _database, _clients
59.       self.socket = clientSocket
60.       self.address = clientAddress[0]
61.       self.id = clientAddress[1]
62.       self.thread = None
63.       self.username = "UNKNOWN"
64.       self.userid = ""
65.       self.admin = False
66.       _clients.append(self)
67.
68.       _logger.log(f"Received a connection from {self.address}, id {self.id}"
    , INFOLOGGINGENABLED)
69.
70.       # Client Login
71.       loginInvalid = True
72.       while loginInvalid:
73.         loginRequestPacket = decode(self.socket.recv(MAXTRANSMISSIONSIZE)) #
     Wait for client login packet TODO: time this out
74.
75.         if loginRequestPacket.type == "CREATEUSER":
76.
77.           usernameExists = _database.userExists(loginRequestPacket.username)

78.
79.           if usernameExists:
80.             userRegistered = RegisterResponsePacket(False, "A user with that
     name already exists")
81.
82.           else:
83.             _database.addUser(loginRequestPacket.username, loginRequestPacke
    t.password)
84.             userRegistered = RegisterResponsePacket(True)
85.             _logger.log(f"Attempted to register user: {loginRequestPacket.us
    ername}. Successful: {userRegistered.valid}", INFOLOGGINGENABLED)
86.
87.           self.socket.send(encode(userRegistered))
88.
89.         elif loginRequestPacket.type == "LOGINREQUEST":
90.           err = "Incorrect username or password"
91.           for client in _clients:
92.             if client.username == loginRequestPacket.username:
93.               valid = False
94.               err = "That user is already logged in"
95.               break
96.
97.           else:
98.             ret = _database.queryLogin(loginRequestPacket.username, loginReq
    uestPacket.password) # Query credentials against database
99.             valid = ret[0]
100.
101.                if not valid:
102.                  _logger.log(f"Invalid login from: {self.address}, id {sel
    f.id} - {err}", INFOLOGGINGENABLED)
103.                  loginResponse = LoginResponsePacket(False, err) # Tell th
    e client the login was invalid
104.                  self.socket.send(encode(loginResponse))
105.
106.                else:
```

```
107.                     _logger.log(f"Valid login from: {self.address}, id {self.
    id}", INFOLOGGINGENABLED)
108.                     loginResponse = LoginResponsePacket(True, userId=ret[1],
    admin=ret[2]) # Tell the client the login was valid
109.                     self.socket.send(encode(loginResponse))
110.                     self.userid = ret[1]
111.                     self.username = loginRequestPacket.username
112.                     self.admin = ret[2]
113.                     loginInvalid = False
114.
115.             readyToListenPacket = decode(self.socket.recv(MAXTRANSMISSIONSI
    ZE)) # Wait until the client is ready to receive packets
116.
117.             # Get as many previous messages as possible that will fit into
    the max transmision size
118.             messagesToSend = []
119.             newMessagesToSend = []
120.             for message in reversed(_database.messages): # reversed as we w
    ant the latest messages
121.               if message.recipientId == 1 or message.senderId == self.useri
    d or message.recipientId == self.userid:
122.                 newMessagesToSend.append(message)
123.                 if len(encode(newMessagesToSend)) >= MAXTRANSMISSIONSIZE -
     200: # We need a buffer of ~200 as when we construct the class the encoded
    size increases
124.                   newMessagesToSend = messagesToSend
125.                   break
126.                 else:
127.                   messagesToSend = newMessagesToSend
128.             newMessageListPacket = MessageListPacket(reversed(messagesToSen
    d)) # Send the client the previous messages
129.             self.socket.send(encode(newMessageListPacket))
130.
131.
132.             newMessage = generateJoinLeaveMessage("joined", self.username)

133.             newMessage = _database.addMessage(newMessage)
134.             announceUserPacket = MessagePacket(newMessage) # Client has joi
    ned message
135.             sendToClients(announceUserPacket)
136.
137.             self.listenerThread = Thread(target=self.ListenForPackets) # St
    art thread to listen for packets from client
138.             self.listenerThread.start()
139.
140.             sendOnlineUsersPacket() # Tell clients a new user has joined
141.
142.           except ConnectionResetError: # Lost connection with client
143.             _logger.log(f"Lost connection with: {self.address}, id {self.id
    }; closing connection", INFOLOGGINGENABLED)
144.
145.             self.socket.close() # Close socket
146.             for i in range(0, len(_clients)):
147.               if _clients[i].id == self.id:
148.                 del _clients[i] # Delete class instance
149.                 break
150.
151.             if self.username != "UNKNOWN":
152.               newMessage = generateJoinLeaveMessage("left", self.username)

153.               newMessage = _database.addMessage(newMessage)
154.               announceUserPacket = MessagePacket(newMessage)
155.               sendToClients(announceUserPacket)
156.               sendOnlineUsersPacket() # Tell clients a user has left
157.
```

```
158.              return # Return from thread
159.            except Exception as err:
160.              reportError(err, _logger)
161.
162.
163.          def ListenForPackets(self):
164.            """
165.            Listens for all communication from client. Should be called async
    ronously in it's own thread.
166.
167.            ToDo:
168.            Break this up?
169.            """
170.            try:
171.              global _clients, _database, _logger
172.              while True:
173.
174.                packet = decode(self.socket.recv(MAXTRANSMISSIONSIZE)) # Wait
     for message from client
175.                if packet.type == "MESSAGE":
176.                  packet.message.timeSent = getDateTime() # Update the messag
    e with the time it was received
177.                  packet.message = _database.addMessage(packet.message)
178.                  sendToClients(packet)
179.                  _logger.log(f"{packet.message.senderName}: {packet.message.
    contents}", INFOLOGGINGENABLED)
180.
181.                elif packet.type == "COMMAND":
182.                  command = packet.command
183.                  args = packet.args
184.                  success = False
185.                  err = ""
186.                  response = ""
187.                  targetClient = self
188.                  _logger.log(f"User {self.username}, {self.id} executed comm
    and {command} with args {args}", INFOLOGGINGENABLED)
189.
190.                  if command == "help":
191.                    success = True
192.                    response = ["!*Available Commands*!",
193.                                ("help","provides a list of available command
    s"),
194.                                ("ping","pings the server"),
195.                                ("whisper <user> <message>","sends a direct m
    essage to <user>"),
196.                                ("markup","displays balsamiq markup syntax")

197.                                ]
198.
199.                  elif command == "markup":
200.                    success = True
201.                    response = ["!*Markup Syntax*!",
202.                                "Formats can be combined and symbols can be e
    scaped with '\\'",
203.                                ("bold", "*example*", "\*example*"),
204.                                ("italic", "_example_", "\_example_"),
205.                                ("strikethrough", "~example~", "\~example~"),

206.                                ("underline", "!example!", "\!example!")
207.                                ]
208.
209.                  elif command == "ping":
210.                    success = True
211.                    response = "Pong!"
212.
```

```
213.                 elif command == "whisper":
214.                   targetName = args[0]
215.                   del args[0]
216.                   for client in _clients:
217.                     if client.username == targetName:
218.                       targetClient = client
219.                       success = True
220.                       message = "_ (Whisper) " + " ".join(args) + "_"
221.                       response = formatUsername(self.username) + message
222.                       newMessage = Message(self.userid, self.username, mess
    age, getDateTime(), targetClient.userid, INFO)
223.                       newMessage = _database.addMessage(newMessage)
224.                       break
225.                   else:
226.                     err = f"Could not find user with name {targetName}"
227.
228.                 else:
229.                   err = "Unrecognised command"
230.
231.                 response = CommandResponsePacket(command, success, err, res
    ponse, getDateTime())
232.                 self.socket.send(encode(response))
233.                 if targetClient != self:
234.                   targetClient.socket.send(encode(response))
235.
236.               elif packet.type == "REQUESTUSERLIST":
237.                 userlist = _database.listUsers()
238.                 self.socket.send(encode(UserListPacket(userlist)))
239.
240.               elif packet.type == "REQUESTUSERINFO":
241.                 userinfo = _database.getUserDetails(packet.user)
242.                 userInfoPacket = UserInfoPacket(userinfo[0], userinfo[1], u
    serinfo[2], userinfo[3])
243.                 self.socket.send(encode(userInfoPacket))
244.
245.               elif packet.type == "REPORTPACKET":
246.                 _database.addReport(packet.messageId, packet.reporterId, pa
    cket.reportReason)
247.                 _logger.log(f"{self.username} registered report: {packet.re
    portReason}", INFOLOGGINGENABLED)
248.
249.               elif packet.type == "EDITMESSAGE":
250.                 _database.editMessage(packet.messageId, packet.newContents)

251.                 i = 0
252.                 oldMessage = None
253.                 for message in _database.messages:
254.                   if message.messageId == packet.messageId:
255.                     oldMessage = _database.messages[i].contents
256.                     _database.messages[i].contents = packet.newContents
257.                     break
258.                   i+=1
259.                 sendToClients(packet) # Tell clients that the message has b
    een edited.
260.                 _logger.log(f"{self.username} edited message from '{oldMess
    age}' to '{packet.newContents}'", INFOLOGGINGENABLED)
261.
262.               elif packet.type == "DELETEMESSAGE":
263.                 _database.deleteMessage(packet.messageId)
264.                 i = 0
265.                 oldMessage = None
266.                 for message in _database.messages:
267.                   if message.messageId == packet.messageId:
268.                     oldMessage = _database.messages[i].contents
269.                     _database.messages[i].contents = "_message deleted_"
```

```
270.                     _database.messages[i].senderId = 1
271.                     _database.messages[i].senderName = "SERVER"
272.                     _database.messages[i].colour = INFO
273.                     break
274.                   i+=1
275.                 sendToClients(packet) # Tell clients that the message has b
    een deleted.
276.                 _logger.log(f"{self.username} deleted message: {oldMessage}
    ", INFOLOGGINGENABLED)
277.
278.               elif packet.type == "SETADMINSTATUS":
279.                 _database.setAdmin(packet.admin, packet.userId)
280.                 _logger.log(f"User id {packet.userId} set to admin: {packet
    .admin}", INFOLOGGINGENABLED)
281.
282.               else:
283.                 _logger.log(f"Unknown packet received: {packet.type}", INFO
    LOGGINGENABLED)
284.
285.           except ConnectionResetError: # Lost connection with client
286.             _logger.log(f"Lost connection with: {self.address}, id {self.id
    }; closing connection", INFOLOGGINGENABLED)
287.
288.             self.socket.close() # Close socket
289.             for i in range(0, len(_clients)):
290.               if _clients[i].id == self.id:
291.                 del _clients[i] # Delete class instance
292.                 break
293.
294.             newMessage = generateJoinLeaveMessage("left", self.username)
295.             newMessage = _database.addMessage(newMessage)
296.             announceUserPacket = MessagePacket(newMessage)
297.             sendToClients(announceUserPacket)
298.             sendOnlineUsersPacket() # Tell clients a user has left
299.
300.           except Exception as err:
301.             reportError(err, _logger)
302.
303.
304.
305.       # Functions
306.
307.       def sendToClients(packet):
308.         """
309.         Sends a packet to every connected client.
310.
311.         Args:
312.           packet (packets.packet): The packet instance to send to the clien
    ts.
313.         """
314.         try:
315.           global _clients
316.           for client in _clients:
317.             client.socket.send(encode(packet))
318.         except ConnectionResetError: # Client has lost connection
319.           pass
320.         except Exception as err:
321.           reportError(err, _logger)
322.
323.
324.       def sendOnlineUsersPacket():
325.         """
326.         Constructs and sends a packet containing all users that are online.

327.         """
```

```
328.          try:
329.            global _clients
330.            users = []
331.            for client in _clients:
332.              users.append(client.username)
333.            users = stringListMergeSort(users)
334.            onlineUsersPacket = OnlineUsersPacket(users)
335.            sendToClients(onlineUsersPacket)
336.          except Exception as err:
337.            reportError(err, _logger)
338.
339.
340.        def __main__():
341.          """
342.          The main body of the program; it all starts here.
343.          Loads database, then listens for connections and assigns them new t
    hreads as they come in.
344.          """
345.          try:
346.            global _database, _logger, INFOLOGGINGENABLED, MAXTRANSMISSIONSIZ
    E
347.            _logger = Logger()
348.
349.            _configManager = ServerConfig("config.json")
350.            INFOLOGGINGENABLED = _configManager.data["infoLoggingEnabled"]
351.            MAXTRANSMISSIONSIZE = _configManager.data["maxTransmissionSize"]

352.
353.            _logger.log("Server started up", INFOLOGGINGENABLED)
354.
355.            _logger.log("Loading Database...", INFOLOGGINGENABLED)
356.            _database = Database(_configManager.data["dbFile"])
357.            _database.loadMessages()
358.            # Create a socket object
359.            serverSocket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

360.
361.            # Get local machine name and assign a port
362.            host = socket.gethostname()
363.            port = _configManager.data["port"]
364.
365.            # Bind to the port
366.            serverSocket.bind((host, port))
367.
368.            # Queue up to 5 requests
369.            serverSocket.listen(5)

370.            _logger.log("Listening for connections...", INFOLOGGINGENABLED)
371.
372.            while True:
373.              # Wait for connections
374.              clientSocket, clientAddress = serverSocket.accept()
375.              newClient = Client(clientSocket, clientAddress)
376.          except Exception as err:
377.            reportError(err, _logger)
378.
379.
380.
381.        if __name__ == "__main__":
382.           __main__()
```

3.1.2 logger.py

```
from threading import *

# Load classes and functions from shared libs
import sys
sys.path.insert(0, '../Libs')
from photonUtilities import *

class Logger:
    """
    A simple logger to log server actions.

    Attributes:
    logQueue (photonUtilities.CircularQueue): The queue used for log write commands in the logWriter() method.
    logThread (threading.Thread): The separate thread started for the logger.
    """
    def __init__(self):
        """" Initialises the logger. """
        self.logQueue = CircularQueue(999)
        self.logThread = Thread(target=self.logWriter)
        self.logThread.start()

    def logWriter(self):
        """
        Writes all strings in the queue sequentially.
        Should be run asynchronously.
        """
        while True:
            if not self.logQueue.isEmpty():
                with open("log.txt", "a+") as logFile:
                    logFile.write(self.logQueue.deQueue())

    def log(self, message, enabled=True):
        """ Appends a string to the queue in the correct format. """
        if enabled:
            message = formatDateTime(getDateTime()) + message
            self.logQueue.enQueue(message + "\n")
            print(message)
```

3.1.3 database.py

```
import sqlite3
from threading import *

# Load classes and functions from shared libs
import sys
sys.path.insert(0, '../Libs')
from photonUtilities import *

class Database:
  """
  Contains all methods relating to reading and writing from the database.

  Attributes:
    roConnection (sqlite3.Connection): The main read only database connection. Used for most get functions.
    roCursor  (sqlite3.Cursor): The cursor for the main read only connection.
    writeQueue (photonUtilities.CircularQueue): The queue used for database write commands in the dbWriter() method.
```

```
    writeThread (threading.Thread): The separate thread started for the data
base writer.
  """
  def __init__(self, file):
    """ Initialises the database by creating a read only connection and star
ting the asyncronous writer function """
    try:
      self.roConnection = sqlite3.connect(f"file:{file}?mode=ro", uri=True)
# Load database from file in read only mode
      self.roCursor = self.roConnection.cursor()
      self.messages = self.loadMessages()
      self.writeQueue = CircularQueue(999)
      self.writeThread = Thread(target=self.dbWriter)
      self.writeThread.start()
    except Exception:
      reportError()


  def dbWriter(self):
    """
    Writes all SQL statements in the queue sequentially as writes to the dat
abase must be done one at a time.
    Should be run asynchronously.
    """
    try:
      while True:
        if not self.writeQueue.isEmpty():
          connection = sqlite3.connect("photon.db")
          cursor = connection.cursor()
          command = self.writeQueue.deQueue()
          cursor.execute(command[0], command[1]) # Execute SQL command
          connection.commit() # Save changes to DB
          cursor.execute("SELECT last_insert_rowid()")
          if len(command) == 4:
            self.messages[command[3]].messageId = cursor.fetchall()[0][0]
          command[2].release()  # Release semaphore flag so the client threa
d can continue
          connection.close()
    except Exception:
      reportError()


  def queryLogin(self, username, password):
    """
    Checks whether supplied login credenitals are valid.

    Args:
      username (string): the username of the user account to check.
      password (string): the password of the user account to check. Should b
e a hash of the password, not the plaintext.

    Returns:
      (bool): True if login successful, False if unsuccessful.
    """
    try:
      self.roCursor.execute("SELECT * FROM User")
      users = self.roCursor.fetchall()
      for user in users: # [0]: id [1]: name [2]: password [3]: admin
        if user[1] == username and user[2] == password:
          if user[3] == 1:
            return (True, user[0], True)
          else:
            return (True, user[0], False)
      return (False)
    except Exception:
```

```
76.       reportError()
77.
78.
79.   def loadMessages(self, count=510): # Load last x messages from database
80.     """
81.     Loads the most recent messages from the database, and stores them global
    ly.
82.
83.     Args:
84.       count (int, optional): the amount of messages to load from the databas
    e.
85.     """
86.     try:
87.         constructedMessages = []
88.         self.roCursor.execute("SELECT * FROM Message limit ? offset (SELECT
    count(*) FROM Message)-?", (str(count), str(count)))
89.         messages = self.roCursor.fetchall()
90.         for message in messages:
91.           self.roCursor.execute("SELECT name FROM User WHERE user_id == ?",
    (str(message[1]),))
92.           username = self.roCursor.fetchall()[0][0]
93.           constructedMessage = Message(messageId=message[0], senderId=messag
    e[1], senderName=username, contents=message[2], timeSent=message[3], recipie
    ntId=message[4], colour=message[5])
94.           constructedMessages.append(constructedMessage)
95.         return constructedMessages
96.     except Exception:
97.         reportError()
98.
99.
100.         def userExists(self, username):
101.           """
102.           Check to see if a user with specific username exists.
103.
104.           Args:
105.             username (string): the username of the user to lookup.
106.
107.           Returns:
108.             (bool): True if the user exists, False if the user does not.
109.           """
110.           self.roCursor.execute("SELECT name FROM User WHERE name == ?", (u
    sername,))
111.           if len(self.roCursor.fetchall()) > 0:
112.             return True
113.           else:
114.             return False
115.
116.
117.         def listUsers(self):
118.           """
119.           Gets a list of all registered users
120.
121.           Returns:
122.             (list of (int, string, bool)): Returns a list of tuples conaini
    ng the userId, username and whether they're admin.
123.
124.           ToDo:
125.             Limit the amount of users that can be fetched at once.
126.           """
127.           connection = sqlite3.connect("file:photon.db?mode=ro", uri=True)

128.           cursor = connection.cursor()
129.           cursor.execute("SELECT user_id, name, admin FROM User WHERE name
    != 'SERVER'")
130.           users = cursor.fetchall()
```

```
131.            return users
132.
133.
134.          def getUserDetails(self, user):
135.            """
136.            Gets the userId, message count, admin and reports for a specific
    user.
137.
138.            Args:
139.              user (string): the username of the user who's info should be fe
    tched.
140.
141.            Returns:
142.              (int, int, bool list of (string, string, string, int)): The det
    ails of the user, corresponding to (user id, message count, whether admin, l
    ist of (reported message, report reason, reporter name, reporter id).
143.            """
144.            try:
145.              connection = sqlite3.connect("file:photon.db?mode=ro", uri=True
    )
146.              cursor = connection.cursor()
147.              cursor.execute("SELECT user_id, admin FROM User WHERE name == ?
    ", (user,))
148.              ret = cursor.fetchall()[0]
149.              userId = ret[0]
150.              admin = bool(ret[1])
151.              cursor.execute("SELECT count(*) FROM Message WHERE sender_id ==
     ?", (userId,))
152.              messageCount = cursor.fetchall()[0][0]
153.              cursor.execute("SELECT * FROM Flag WHERE reportedUser_id == ?",
     (userId,))
154.              flagged = cursor.fetchall()
155.              flags = []
156.              for flag in flagged:
157.                cursor.execute("SELECT contents FROM Message WHERE message_id
     == ?", (flag[2],))
158.                message = cursor.fetchall()[0][0]
159.                cursor.execute("SELECT name FROM User WHERE user_id == ?", (f
    lag[3],))
160.                reporterName = cursor.fetchall()[0][0]
161.                flags.append((message, flag[4], reporterName, flag[3]))
162.
163.              return (userId, messageCount, admin, flags)
164.            except Exception:
165.              reportError()
166.
167.          def addUser(self, username, password):
168.            """
169.            Creates a new user entry in the database.
170.
171.            Args:
172.              username (string): the username of the account to create.
173.              password (string): the password of the account to create -
     this should be hashed.
174.            """
175.            semaphore = Semaphore(value=0) # Create a semaphore to be used to
     signal once the database write has been completed
176.            self.writeQueue.enQueue(("INSERT into User(name, password) values
     (?, ?)", (username, password), semaphore))
177.            semaphore.acquire() # Wait until semaphore has been released IE h
    as db write is complete
178.
179.
180.          def addMessage(self, message):
181.            """
```

```
182.            Creates a new message entry in the database.
183.
184.            Args:
185.              message (Message): The instance of message class containing the
    information that needs to be written.
186.
187.            Returns:
188.
189.            """
190.            semaphore = Semaphore(value=0) # Create a semaphore to be used to
    tell once the database write has been completed
191.            self.messages.append(message)
192.            messageIndex = len(self.messages) - 1
193.            self.writeQueue.enQueue(("INSERT into Message(sender_id, contents
    , timeSent, recipient_id, colour) values (?,?,?,?,?)", (str(message.senderId
    ), message.contents, message.timeSent, message.recipientId, message.colour),
     semaphore, messageIndex))
194.            semaphore.acquire() # Wait until semaphore has been released IE h
    as db write is complete
195.            return self.messages[messageIndex] # Message object will have bee
    n updated with it's ID by the DB writer
196.
197.          def addReport(self, messageId, reporterId, reportReason):
198.            connection = sqlite3.connect("file:photon.db?mode=ro", uri=True)

199.            cursor = connection.cursor()
200.            cursor.execute("SELECT sender_id FROM Message where message_id ==
    ?", (messageId,))
201.            reportedUserId = cursor.fetchall()[0][0]
202.            semaphore = Semaphore(value=0) # Create a semaphore to be used to
    tell once the database write has been completed
203.            self.writeQueue.enQueue(("INSERT into Flag(reportedUser_id, messa
    ge_id, reporter_id, reportReason) values (?,?,?,?)", (reportedUserId, messag
    eId, reporterId, reportReason), semaphore))
204.            semaphore.acquire() # Wait until semaphore has been released IE h
    as db write is complete
205.
206.          def editMessage(self, messageId, newContent):
207.            semaphore = Semaphore(value=0) # Create a semaphore to be used to
    tell once the database write has been completed
208.            self.writeQueue.enQueue(("UPDATE Message SET contents=? WHERE mes
    sage_id=?", (newContent, messageId), semaphore))
209.            semaphore.acquire() # Wait until semaphore has been released IE h
    as db write is complete
210.
211.          def deleteMessage(self, messageId):
212.            semaphore = Semaphore(value=0) # Create a semaphore to be used to
    tell once the database write has been completed
213.            self.writeQueue.enQueue(("UPDATE Message SET contents=?,sender_id
    =?,colour=? WHERE message_id=?", ("_message deleted_", 1, INFO, messageId),
    semaphore))
214.            semaphore.acquire() # Wait until semaphore has been released IE h
    as db write is complete
215.
216.          def setAdmin(self, admin, userId):
217.            semaphore = Semaphore(value=0) # Create a semaphore to be used to
    tell once the database write has been completed
218.            self.writeQueue.enQueue(("UPDATE User SET admin=? WHERE user_id=?
    ", (admin, userId), semaphore))
219.            semaphore.acquire() # Wait until semaphore has been released IE h
    as db write is complete
```

3.1.4 config.json

```
{
    "dbFile": "photon.db",
    "infoLoggingEnabled": true,
    "maxTransmissionSize": 40960,
    "port": 9998
}
```

3.2 Client

3.2.1 main.py

```
# Main Client File

import sys
import socket
import pickle
from threading import Thread
import atexit
import os
import cgi
import time
import datetime
import re

from PyQt5.QtCore import pyqtSlot, pyqtSignal
from PyQt5.QtWidgets import QApplication, QMainWindow, QDialog, QMessageBox, QWidget, QFormLayout, QScrollArea, QTableWidgetItem
from PyQt5.QtGui import QColor
from PyQt5.uic import loadUi

# Load classes and functions from shared libs
import sys
sys.path.insert(0, '../Libs')
from packets import *
from photonUtilities import *
from configManager import *


# Global vars
_app = None
_mainGui = None
_serverSocket = None
_username = ""
_userId = None
_admin = False

NONPRINTINGCHAR = '\u200B' # Used to replace a character in a string whilst keeping indexes the same
MAXTRANSMISSIONSIZE = None
COMMANDCHAR = None
DEBUG = None


# Classes

class MainWindow(QMainWindow):
  """
  GUI Class for main window. Inherits QMainWindow.
```

```
47.
48.   Elements:
49.     centralWidget (QWidget): Main area containg window elements.
50.     adminSettinsButton (QPushButton): Button to open admin settings.
51.     messageInput (QLineEdit): Input area for sending messages.
52.     messageInputButton (QPushButton): Send button for messages.
53.     messageScrollArea (QScrollArea): Window section containing message eleme
    nts.
54.     messageWidget (QWidget): Widget to contain message elements.
55.     userCountLabel (QLabel): Displays count of online users.
56.     userListBox (QTextEdit): Contains all online users.
57.     usernameLabel (QLabel): Displays username.
58.
59.   Properties:
60.     writeSignal (QSignal): Signal to trigger message creation.
61.     usersChangedSignal (QSignal): Signal to update online users.
62.
63.   ToDo:
64.     Seperate this class out!
65.   """
66.
67.   # Signals for updating the GUI
68.   writeSignal = pyqtSignal(Message)
69.   usersChangedSignal = pyqtSignal(list)
70.   updateMessageSignal = pyqtSignal(int, str)
71.   deleteMessageSignal = pyqtSignal(int)
72.
73.   def __init__(self, *args):
74.     """ Initialises the UI and connects all signals and button clicks. """
75.     try:
76.       super().__init__(*args)
77.       loadUi("mainWindow.ui", self)
78.       self.messageInputButton.clicked.connect(self.onSendClick)
79.       self.messageInput.returnPressed.connect(self.onSendClick)
80.       # Point the signals to the corresponding functions
81.       self.writeSignal.connect(self.WriteLine)
82.       self.usersChangedSignal.connect(self.UpdateConnectedUsers)
83.       self.updateMessageSignal.connect(self.updateMessageContents)
84.       self.deleteMessageSignal.connect(self.deleteMessage)
85.       self.messageWidget.setLayout(self.messageLayout)
86.
87.     except Exception:
88.       ReportError()
89.
90.
91.   def setUsername(self, username):
92.     """
93.     Sets the username label text.
94.
95.     Args:
96.       username (string): The username to set to.
97.     """
98.     try:
99.       self.usernameLabel.setText(f"Logged in as {username}")
100.          except Exception:
101.            reportError()
102.
103.
104.       def postLogin(self):
105.         """ Setup for after login is completed. """
106.         if not _admin:
107.             self.adminSettingsButton.hide()
108.         else:
109.             self.adminSettingsButton.clicked.connect(self.openAdminSettin
    gs)
```

```
110.
111.         def openAdminSettings(self):
112.           """ Opens the admin settings UI. """
113.           try:
114.             self.adminSettings = AdminSettingsWindow(self)
115.             self.adminSettings.show()
116.           except Exception:
117.             reportError()
118.
119.
120.         def closeEvent(self, event):
121.           """ When main window closed, tidy up loose ends and exit. """
122.           try:
123.             # Not using onProgramExit() as it caused the program to hang wh
    en the UI is created
124.             global _serverSocket
125.             debugPrint("Window closed: Force closing all threads and server
     socket", DEBUG)
126.             _serverSocket.close()
127.             os._exit(1)
128.           except Exception:
129.             reportError()
130.
131.
132.         def WriteLine(self, message):
133.           """
134.           Formats message, creates a new element for it and displays it pro
    perly.
135.
136.           Args:
137.             message (Message): The message to display
138.           """
139.           rawMessage = message.contents
140.           try:
141.             newWidget = MessageWidget(message=message) # Create a new messa
    ge widget
142.             newWidget.updateText()
143.             rowCount = self.messageLayout.rowCount() # Get the amount of ro
    ws in the message container
144.             self.messageLayout.setWidget(rowCount, QFormLayout.LabelRole, n
    ewWidget) # Append the new message widget to the end of the container
145.             newWidget.setFixedWidth(self.messageScrollArea.width() -
     10) # Set widget width to match the parent width
146.
147.             debugPrint(rawMessage, DEBUG)
148.             _app.alert(_mainGui, 1000) # Flash the taskbar icon for 1 secon
    d
149.           except Exception:
150.             reportError()
151.
152.
153.         def UpdateConnectedUsers(self, userList):
154.           """
155.           Update connected users GUI elements.
156.
157.           Args:
158.             userList (list of string): List of online users.
159.           """
160.           userCount = len(userList)
161.           self.userCountLabel.setText(f"Users Online: {userCount}")
162.
163.           self.userListBox.clear()
164.           for user in userList:
165.             self.userListBox.insertHtml(f"{user} <br>")
166.
```

```
167.
168.         def onSendClick(self):
169.           """ When the send button is pressed, take input, parse it and the
    n send it. """
170.           try:
171.             text = self.messageInput.text()
172.             if not text.isspace() and text != "":
173.               if text[0] == COMMANDCHAR:
174.                 ParseCommand(text)
175.               else:
176.                 SendMessage(text)
177.               self.messageInput.setText("")
178.           except Exception:
179.             reportError()
180.
181.
182.         def resizeEvent(self, event):
183.           """ Overrides the window resize event. Updates widget sizes. """

184.           width = self.messageScrollArea.width() - 10
185.           messageWidgets = (self.messageLayout.itemAt(i) for i in range(sel
    f.messageLayout.count())) # Get a list of widgets in layout
186.
187.           for layoutItem in messageWidgets:
188.             layoutItem.widget().setFixedWidth(width)
189.
190.
191.         def updateMessageContents(self, messageId, contents):
192.           messageWidgets = (self.messageLayout.itemAt(i) for i in range(sel
    f.messageLayout.count())) # Get a list of widgets in layout
193.
194.           for layoutItem in messageWidgets:
195.             if layoutItem.widget().message.messageId == messageId:
196.               layoutItem.widget().message.contents = contents
197.               layoutItem.widget().updateText()
198.
199.         def deleteMessage(self, messageId):
200.           messageWidgets = (self.messageLayout.itemAt(i) for i in range(sel
    f.messageLayout.count())) # Get a list of widgets in layout
201.
202.           for layoutItem in messageWidgets:
203.             if layoutItem.widget().message.messageId == messageId:
204.               layoutItem.widget().message.contents = "_message deleted_"
205.               layoutItem.widget().message.senderId = 1
206.               layoutItem.widget().message.senderName = "SERVER"
207.               layoutItem.widget().message.colour = INFO
208.               layoutItem.widget().updateText()
209.
210.
211.
212.       class MessageWidget(QWidget):
213.         """ Gui Class for each message. """
214.         def __init__(self, parent=None, message=""):
215.             super().__init__(parent)
216.             loadUi("message.ui", self)
217.             self.messageOptionBtn.clicked.connect(lambda: self.openMessageO
    ptions())
218.             self.message = message
219.             self.setMouseTracking(1)
220.             hoverColour = QColor("#f0f0f0")
221.             self.hoverPalette = self.palette()
222.             self.hoverPalette.setColor(self.backgroundRole(), hoverColour)

223.             baseColour = QColor("#ffffff")
224.             self.basePalette = self.palette()
```

```
225.             self.basePalette.setColor(self.backgroundRole(), baseColour)
226.
227.         def openMessageOptions(self):
228.           """ Opens UI for managing messages. """
229.           try:
230.             self.messageOptions = MessageOptions(self, message=self.message
    )
231.             self.messageOptions.show()
232.           except Exception:
233.             reportError()
234.
235.         def updateText(self):
236.             self.timeLabel.setText(self.message.timeSent)
237.             self.usernameLabel.setText(formatUsername(self.message.senderNa
    me))
238.             self.messageLabel.setText(formatTextForDisplay(self.message.con
    tents, self.message.colour))
239.
240.         def enterEvent(self, event):
241.           self.setPalette(self.hoverPalette)
242.
243.         def leaveEvent(self, event):
244.           self.setPalette(self.basePalette)
245.
246.       class MessageOptions(QDialog):
247.         """
248.         GUI Class for managing messages
249.
250.         Args:
251.           timestamp (string): Time that the message was sent.
252.           user (string): Username of user who sent the message.
253.           message (string): The contents of the message being managed.
254.         """
255.         def __init__(self, *args, message=""):
256.           try:
257.             super().__init__(*args)
258.             loadUi("messageOptions.ui", self)
259.             self.reportButton.clicked.connect(lambda: self.sendReport())
260.             self.editButton.clicked.connect(lambda: self.editMessageContent
    s())
261.             self.deleteButton.clicked.connect(lambda: self.deleteMessage())

262.             self.message = message
263.             self.timeLabel.setText(message.timeSent)
264.             self.usernameLabel.setText(formatUsername(message.senderName))

265.             self.editMessage.setText(message.contents)
266.
267.             # Users can only edit and delete their own messages. Admins can
     delete any message. You cannot report your own or a server message. Local m
    essages cannot be modified.
268.             if self.message.senderId != _userId or self.message.senderId ==
     "":
269.               self.editMessage.setEnabled(False)
270.               self.editButton.setEnabled(False)
271.               if not _admin or self.message.senderId == "":
272.                 self.deleteButton.setEnabled(False)
273.
274.             if self.message.senderId == 1 or self.message.senderId == _user
    Id or self.message.senderId == "":
275.               self.reportButton.setEnabled(False)
276.               self.reportReason.setEnabled(False)
277.
278.           except Exception:
279.             reportError()
```

```
280.
281.          def sendReport(self):
282.            reason = self.reportReason.text()
283.            if len(reason) < 1:
284.              errNotifier = QMessageBox()
285.              errNotifier.setIcon(QMessageBox.Critical)
286.              errNotifier.setText("You must supply a reason when reporting a
    message.")
287.              errNotifier.setWindowTitle("Report Error")
288.              errNotifier.exec_()
289.            else:
290.              reportPacket = ReportPacket(self.message.messageId, _userId, re
    ason)
291.              _serverSocket.send(encode(reportPacket))
292.              successNotifier = QMessageBox()
293.              successNotifier.setIcon(QMessageBox.Information)
294.              successNotifier.setText("Successfully reported message.")
295.              successNotifier.setWindowTitle("Report Result")
296.              successNotifier.exec_()
297.
298.          def editMessageContents(self):
299.            editMessagePacket = EditMessagePacket(self.message.messageId, sel
    f.editMessage.text())
300.            _serverSocket.send(encode(editMessagePacket))
301.            successNotifier = QMessageBox()
302.            successNotifier.setIcon(QMessageBox.Information)
303.            successNotifier.setText("Successfully edited message.")
304.            successNotifier.setWindowTitle("Edit Result")
305.            successNotifier.exec_()
306.
307.          def deleteMessage(self):
308.            deleteMessagePacket = DeleteMessagePacket(self.message.messageId)

309.            _serverSocket.send(encode(deleteMessagePacket))
310.            successNotifier = QMessageBox()
311.            successNotifier.setIcon(QMessageBox.Information)
312.            successNotifier.setText("Successfully deleted message.")
313.            successNotifier.setWindowTitle("Delete Result")
314.            successNotifier.exec_()
315.            self.close()
316.
317.
318.        class AdminSettingsWindow(QDialog):
319.          """ GUI Class for the admin window. """
320.          def __init__(self, *args):
321.              super().__init__(*args)
322.              loadUi("adminSettings.ui", self)
323.              self.userListComboBox.currentIndexChanged.connect(self.ComboBox
    Updated)
324.              self.toggleAdminStatusBtn.clicked.connect(self.toggleAdminStatu
    s)
325.              requestUserListPacket = Packet("REQUESTUSERLIST")
326.              _serverSocket.send(encode(requestUserListPacket))
327.              self.selectedUserAdmin = None
328.              self.selectedUserId = None
329.
330.          def UserListReceived(self, userList):
331.            """
332.            Populates the userList dropdown.
333.
334.            Args:
335.              userList (list of string): The list of users.
336.            """
337.            for user in userList:
338.              self.userListComboBox.addItem(user[1])
```

```
339.
340.         def ComboBoxUpdated(self):
341.           """ Event for when a new user is selected from the combobox. Fetc
    hes information about that user. """
342.           requestUserInfoPacket = RequestUserInfoPacket(self.userListComboB
    ox.currentText())
343.           _serverSocket.send(encode(requestUserInfoPacket))
344.
345.         def UpdateUserInfo(self, userId, messageCount, admin, flags):
346.           """
347.           Updates the user specific info sections.
348.
349.           Args:
350.             userId (int): The id of the selected user.
351.             messageCount (int): The amount of messages sent by that user.
352.             admin (bool): Whether the user is admin.
353.             flags  (list of (string, string, string, int)): The report deta
    ils of that user. Corresponding to list of (reported message, report reason,
     reporter name, reporter id)
354.           """
355.           self.userIdLabel.setText(str(userId))
356.           self.selectedUserId = str(userId)
357.           self.messageCountLabel.setText(str(messageCount))
358.           self.reportCountLabel.setText(str(len(flags)))
359.
360.           rowPosition = self.reportTable.rowCount()
361.           for row in range(0, rowPosition):
362.             self.reportTable.removeRow(1)
363.
364.           self.selectedUserAdmin = admin
365.           if admin:
366.             self.toggleAdminStatusBtn.setText("Demote from Admin")
367.           else:
368.             self.toggleAdminStatusBtn.setText("Promote to Admin")
369.           self.toggleAdminStatusBtn.setEnabled(True)
370.
371.           i = 1
372.           for flag in flags:
373.             self.reportTable.insertRow(i)
374.             self.reportTable.setItem(i, 0, QTableWidgetItem(str(flag[0])))
375.             self.reportTable.setItem(i, 1, QTableWidgetItem(str(flag[1])))
376.             self.reportTable.setItem(i, 2, QTableWidgetItem(f"{flag[2]} (ID
    : {flag[3]})"))
377.             i += 1
378.
379.         def toggleAdminStatus(self):
380.           adminStatusPacket = SetAdminStatusPacket(not self.selectedUserAdm
    in, self.selectedUserId)
381.           _serverSocket.send(encode(adminStatusPacket))
382.           self.ComboBoxUpdated()
383.
384.       class LoginWindow(QDialog):
385.         """ GUI Class for login window """
386.         def __init__(self, *args):
387.           try:
388.             super().__init__(*args)
389.             loadUi("login.ui", self)
390.             self.loginButton.clicked.connect(self.onLoginClick)
391.             self.newAccButton.clicked.connect(self.openRegisterWindow)
392.             self.usernameInput.returnPressed.connect(self.onLoginClick)
393.           except Exception:
394.             reportError()
395.
```

```
396.
397.         def openRegisterWindow(self):
398.           """ Opens register window GUI """
399.           try:
400.             registerGui = RegisterWindow()
401.             registerGui.exec_()
402.           except Exception:
403.             reportError()
404.
405.
406.         def onLoginClick(self):
407.           """ Called on login button click. Basic input validation. """
408.           try:
409.             global _username
410.             username = self.usernameInput.text()
411.             password = self.passwordInput.text()
412.
413.             if not username.isspace() and username != "":
414.               if not password.isspace() and password != "":
415.                 Password = password
416.                 self.Login(username, password)
417.
418.               else:
419.                 self.errLabel.setText("Passwords must not consist of whites
    pace only")
420.
421.             else:
422.               self.errLabel.setText("Usernames must not consist of whitespa
    ce only")
423.           except Exception:
424.             reportError()
425.
426.
427.         def Login(self, username, password):
428.           """
429.           Attempts to login user to server.
430.
431.           Args:
432.             username (string): Username to login with.
433.             password (string): Password to login with, not yet hashed.
434.           """
435.           try:
436.             global _serverSocket, _username, _userId, _admin
437.             password = hashString(password)
438.             loginRequest = LoginRequestPacket(username, password)
439.             _serverSocket.send(encode(loginRequest))
440.
441.             loginResponsePacket = decode(_serverSocket.recv(MAXTRANSMISSION
    SIZE))
442.
443.             if loginResponsePacket.type != "LOGINRESPONSE":
444.               self.Login(username, password) # Occasionally a left-
    over packet can make it's way here - if so we'll just try again
445.               return
446.
447.             if loginResponsePacket.valid:
448.               self.close()
449.               _username = username
450.               _userId = loginResponsePacket.id
451.               _admin = loginResponsePacket.admin
452.
453.             else:
454.               self.errLabel.setText(loginResponsePacket.err)
455.           except Exception:
456.             reportError()
```

```
457.
458.
459.       class RegisterWindow(QDialog):
460.         """ GUI Class for register window """
461.         def __init__(self, *args):
462.           try:
463.             super().__init__(*args)
464.             loadUi("register.ui", self)
465.             self.registerButton.clicked.connect(self.validateInputs)
466.           except Exception:
467.             reportError()
468.
469.         def validateInputs(self):
470.           """ Called on register button click. Basic input validation. """

471.           try:
472.             username = self.usernameInput.text()
473.             password1 = self.passwordInput1.text()
474.             password2 = self.passwordInput2.text()
475.
476.             if not username.isspace() and username != "" and len(username)
    < 33:
477.               if password1 == password2:
478.                 if not password1.isspace() and password1 != "":
479.                   self.register(username, password1)
480.                 else:
481.                   self.errLabel.setText("Passwords must not consist of whit
    espace only")
482.               else:
483.                 self.errLabel.setText("Passwords must match")
484.             else:
485.               self.errLabel.setText("Usernames must not consist of whitespa
    ce only, and be\nless than 33 chars long")
486.           except Exception:
487.             reportError()
488.
489.         def register(self, username, password):
490.           """
491.           Attempts to register user with server.
492.
493.           Args:
494.             username (string): Username to register.
495.             password (string): Password to register, not yet hashed.
496.           """
497.           try:
498.             password = hashString(password)
499.             registerPacket = RegisterPacket(username, password)
500.             global _serverSocket
501.             _serverSocket.send(encode(registerPacket))
502.             registerResponse = decode(_serverSocket.recv(MAXTRANSMISSIONSIZ
    E)) # Wait for user creation packet response
503.             if registerResponse.valid:
504.               successNotifier = QMessageBox()
505.               successNotifier.setIcon(QMessageBox.Information)
506.               successNotifier.setText(f"Successfully registered user '{user
    name}'")
507.               successNotifier.setWindowTitle("Account creation successful")

508.               successNotifier.exec_()
509.               self.close()
510.             else:
511.               self.errLabel.setText(registerResponse.err)
512.           except Exception:
513.             reportError()
514.
```

```
515.
516.
517.
518.        # Functions
519.        """
520.        Constructs a message object and sends it to the server.
521.
522.        Args:
523.          message (string): The message contents.
524.        """
525.        def SendMessage(message):
526.          try:
527.              global _serverSocket, _username
528.              debugPrint(_userId, DEBUG)
529.              newMessage = Message(_userId, _username, message)
530.              newMessagePacket = MessagePacket(newMessage)
531.              _serverSocket.send(encode(newMessagePacket))
532.
533.          except Exception:
534.             reportError()
535.
536.        """
537.        Formats a command string into arguments and sends it to the server.
538.
539.        Args:
540.          command (string): The raw command text before formatting.
541.        """
542.        def ParseCommand(command):
543.          try:
544.            global _serverSocket
545.            command = command[1:] # Strip command char
546.            args = command.split(" ")
547.            command = args[0]
548.            del args[0]
549.            commandPacket = CommandPacket(command, args)
550.            _serverSocket.send(encode(commandPacket))
551.
552.          except Exception:
553.            reportError()
554.
555.        """
556.        Ensures a message is in the correct format and then passes it to the
    GUI to display.
557.
558.        Args:
559.          message (string, optional): The string to convert to message object
    .
560.          message (Message, optional): The message object to pass to the GUI.

561.        """
562.        def printMessage(message):
563.          if type(message) is str:
564.            message = Message(contents=message)
565.
566.          _mainGui.writeSignal.emit(message)
567.
568.
569.        def onProgramExit():
570.          try:
571.            global _serverSocket
572.            debugPrint("Window closed: Force closing all threads and server s
    ocket", DEBUG)
573.            _serverSocket.close()
574.            os._exit(1)
575.          except Exception:
```

```
576.             reportError()
577.
578.
579.       def ListenForPackets(server):
580.         try:
581.           global _serverSocket, _mainGui
582.
583.           readyToListen = Packet("READYTOLISTEN") # Tell the server we are
    ready to listen using generic packet
584.           _serverSocket.send(encode(readyToListen))
585.
586.           while True:
587.             packet = decode(server.recv(MAXTRANSMISSIONSIZE))
588.
589.             if packet.type == "MESSAGELIST":
590.               for message in packet.messageList:
591.                 printMessage(message)
592.
593.             elif packet.type == "MESSAGE":
594.               formatMessage(packet)
595.
596.             elif packet.type == "ONLINEUSERS":
597.               _mainGui.usersChangedSignal.emit(packet.userList)
598.
599.             elif packet.type == "COMMANDRESPONSE":
600.               if packet.success:
601.                 if packet.command == "help":
602.                   printMessage(packet.response[0])
603.                   for i in range(1, len(packet.response)):
604.                     printMessage(f" -
     *{packet.response[i][0]}* : {packet.response[i][1]}")
605.
606.                 if packet.command == "markup":
607.                   printMessage(packet.response[0])
608.                   printMessage(packet.response[1])
609.                   for i in range(2, len(packet.response)):
610.                     printMessage(f" -
     {packet.response[i][0]}, {packet.response[i][1]} : {packet.response[i][2]}"
    )
611.
612.                 elif packet.command == "ping":
613.                   printMessage(Message(contents=packet.response, colour=INF
    O))
614.
615.                 elif packet.command == "whisper":
616.                   printMessage(Message(contents=formatDateTime(packet.timeS
    ent) + packet.response, colour=INFO))
617.
618.               else:
619.                 printMessage(Message(contents=f"Error executing command '{p
    acket.command}' - {packet.err}", colour=COMMANDERROR))
620.
621.
622.             elif packet.type == "USERLIST":
623.               _mainGui.adminSettings.UserListReceived(packet.userList)
624.
625.
626.             elif packet.type == "USERINFO":
627.               _mainGui.adminSettings.UpdateUserInfo(packet.id, packet.messa
    geCount, packet.admin, packet.flags)
628.
629.             elif packet.type == "EDITMESSAGE":
630.               _mainGui.updateMessageSignal.emit(packet.messageId, packet.ne
    wContents)
631.
```

```
632.                elif packet.type == "DELETEMESSAGE":
633.                  _mainGui.deleteMessageSignal.emit(packet.messageId)
634.
635.                else:
636.                    print(f"Unknown packet received: {packet.type}")
637.
638.
639.
640.          except Exception:
641.            reportError()
642.
643.
644.        def formatMessage(packet):
645.          try:
646.            printMessage(packet.message)
647.          except Exception:
648.              reportError()
649.
650.
651.        def formatTextForDisplay(message, colour):
652.          try:
653.            message = cgi.escape(message) # Escape html code; sanitise input

654.            # Replace balsamiq chars in pairs with html
655.            message = formatBalsmaiq(message, "*", "b")
656.            message = formatBalsmaiq(message, "_", "i")
657.            message = formatBalsmaiq(message, "~", "s")
658.            message = formatBalsmaiq(message, "!", "u")
659.
660.            message = f"<font color='{colour}'> {message} </font>"
661.
662.            return message
663.          except Exception:
664.             reportError()
665.
666.        def inverseFormatTextForDisplay(message):
667.            try:
668.              colour = re.search("<font color='(.+?)'>", message).group(1) #
    Use regex to extract the hex colour code of the message
669.              # Replace html pairs with balsamiq
670.              message = re.sub("<b>|</b>", "*", message)
671.              message = re.sub("<i>|</i>", "_", message)
672.              message = re.sub("<s>|</s>", "~", message)
673.              message = re.sub("<u>|</u>", "!", message)
674.              message = re.sub('<[^<]+?>', '', message) # strip remaining htm
    l code
675.
676.              return (message, colour)
677.            except Exception:
678.              reportError()
679.
680.
681.        def formatBalsmaiq(message, specialChar, tag):
682.          try:
683.            global NONPRINTINGCHAR
684.            charInstances = []
685.            i = 0
686.            while True: # Locate all instances of special char within message

687.              charInstance = message.find(specialChar, i)
688.              if charInstance == -1:
689.                break
690.              else:
691.                charInstances.append(charInstance)
692.                i = charInstance + 1
```

```
693.            message = list(message) # Convert to list so we can substitute ch
    ars by index
694.            i = 0
695.            for charInstance in charInstances: # Ensure character is not esca
    ped by '\'
696.              if message[charInstance - 1] == "\\":
697.                message[charInstance - 1] = NONPRINTINGCHAR
698.                del charInstances[i]
699.              else:
700.                i+= 1
701.
702.            if len(charInstances) % 2 != 0: # Ignore special chars without a
    pair
703.              charInstances = charInstances[:-1]
704.
705.            for i in range(0, len(charInstances), 2): # Replace pairs of bals
    amiq with html code
706.              message[charInstances[i]] = f"<{tag}>"
707.              message[charInstances[i + 1]] = f"</{tag}>"
708.
709.            message = "".join(message) # Convert back to string
710.            return message
711.          except Exception:
712.              reportError()
713.
714.
715.        def __main__():
716.          try:
717.            global _serverSocket, _username, MAXTRANSMISSIONSIZE, COMMANDCHAR
    , DEBUG
718.
719.            _configManager = ClientConfig("config.json")
720.            MAXTRANSMISSIONSIZE = _configManager.data["maxTransmissionSize"]

721.            COMMANDCHAR = _configManager.data["commandChar"]
722.            DEBUG = _configManager.data["debug"]
723.
724.            atexit.register(onProgramExit)
725.
726.            # Print PyQt 'silent' errors
727.            sys._excepthook = sys.excepthook
728.            def exception_hook(exctype, value, traceback):
729.              print(exctype, value, traceback)
730.              sys._excepthook(exctype, value, traceback)
731.              sys.exit(1)
732.            sys.excepthook = exception_hook
733.
734.            # Create a socket object
735.            _serverSocket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

736.
737.            # Get local machine name and assign a port
738.            host = socket.gethostname()
739.            port = _configManager.data["port"]
740.
741.            # Connect to hostname on the port.
742.            _serverSocket.connect((host, port))
743.
744.            # Display UI
745.            global GuiDone, _app, _mainGui
746.            _app = QApplication(sys.argv)
747.            _mainGui = MainWindow()
748.            loginGui = LoginWindow()
749.            loginGui.exec_()
750.            _mainGui.postLogin()
```

```
751.            if _username == "": # Window was closed without an input
752.              os._exit(1)
753.            _mainGui.setUsername(_username)
754.            _mainGui.show()
755.
756.            # Start listener thread for server responses
757.            listenerThread = Thread(target=ListenForPackets, args=(_serverSoc
    ket,))
758.            listenerThread.start()
759.
760.            sys.exit(_app.exec_())
761.
762.          except Exception:
763.              reportError()
764.
765.
766.        if __name__ == "__main__":
767.            __main__()
```

3.2.2 adminSettings.ui

```
1.  <?xml version="1.0" encoding="UTF-8"?>
2.  <ui version="4.0">
3.   <class>AdminSettings</class>
4.   <widget class="QWidget" name="AdminSettings">
5.    <property name="geometry">
6.     <rect>
7.      <x>0</x>
8.      <y>0</y>
9.      <width>400</width>
10.     <height>397</height>
11.    </rect>
12.   </property>
13.   <property name="windowTitle">
14.    <string>Admin Settings</string>
15.   </property>
16.   <layout class="QFormLayout" name="formLayout">
17.    <item row="0" column="0">
18.     <widget class="QLabel" name="label">
19.      <property name="text">
20.       <string>Selected User</string>
21.      </property>
22.     </widget>
23.    </item>
24.    <item row="1" column="0">
25.     <widget class="QComboBox" name="userListComboBox"/>
26.    </item>
27.    <item row="2" column="0" colspan="2">
28.     <widget class="Line" name="line">
29.      <property name="orientation">
30.       <enum>Qt::Horizontal</enum>
31.      </property>
32.     </widget>
33.    </item>
34.    <item row="3" column="0">
35.     <widget class="QLabel" name="label_2">
36.      <property name="text">
37.       <string>Actions</string>
38.      </property>
39.     </widget>
40.    </item>
41.    <item row="4" column="0">
42.     <widget class="QPushButton" name="toggleAdminStatusBtn">
43.      <property name="enabled">
```

```
      <bool>false</bool>
     </property>
     <property name="text">
      <string>Promote to Admin</string>
     </property>
    </widget>
   </item>
   <item row="5" column="0" colspan="2">
    <widget class="Line" name="line_2">
     <property name="orientation">
      <enum>Qt::Horizontal</enum>
     </property>
    </widget>
   </item>
   <item row="6" column="0">
    <widget class="QLabel" name="label_3">
     <property name="text">
      <string>User Info</string>
     </property>
    </widget>
   </item>
   <item row="7" column="0">
    <widget class="QLabel" name="label_5">
     <property name="text">
      <string>User Id:</string>
     </property>
    </widget>
   </item>
   <item row="7" column="1">
    <widget class="QLabel" name="userIdLabel">
     <property name="text">
      <string>0</string>
     </property>
    </widget>
   </item>
   <item row="8" column="0">
    <widget class="QLabel" name="label_4">
     <property name="text">
      <string>Messages Sent:</string>
     </property>
    </widget>
   </item>
   <item row="8" column="1">
    <widget class="QLabel" name="messageCountLabel">
     <property name="text">
      <string>0</string>
     </property>
    </widget>
   </item>
   <item row="9" column="0">
    <widget class="QLabel" name="label_6">
     <property name="text">
      <string>Times Reported:</string>
     </property>
    </widget>
   </item>
          <item row="9" column="1">
           <widget class="QLabel" name="reportCountLabel">
            <property name="text">
             <string>0</string>
            </property>
           </widget>
          </item>
          <item row="10" column="0" colspan="2">
           <widget class="Line" name="line_3">
```

```
109.             <property name="orientation">
110.              <enum>Qt::Horizontal</enum>
111.             </property>
112.            </widget>
113.           </item>
114.           <item row="11" column="0">
115.            <widget class="QLabel" name="label_7">
116.             <property name="text">
117.              <string>Reports</string>
118.             </property>
119.            </widget>
120.           </item>
121.           <item row="12" column="0" colspan="2">
122.            <widget class="QTableWidget" name="reportTable">
123.             <property name="editTriggers">
124.              <set>QAbstractItemView::NoEditTriggers</set>
125.             </property>
126.             <property name="dragEnabled">
127.              <bool>true</bool>
128.             </property>
129.             <property name="alternatingRowColors">
130.              <bool>true</bool>
131.             </property>
132.             <property name="rowCount">
133.              <number>1</number>
134.             </property>
135.             <property name="columnCount">
136.              <number>3</number>
137.             </property>
138.             <attribute name="horizontalHeaderVisible">
139.              <bool>true</bool>
140.             </attribute>
141.             <row/>
142.             <column/>
143.             <column/>
144.             <column/>
145.             <item row="0" column="0">
146.              <property name="text">
147.               <string notr="true">Message</string>
148.              </property>
149.              <property name="font">
150.               <font>
151.                <weight>75</weight>
152.                <bold>true</bold>
153.               </font>
154.              </property>
155.             </item>
156.             <item row="0" column="1">
157.              <property name="text">
158.               <string>Report</string>
159.              </property>
160.              <property name="font">
161.               <font>
162.                <weight>75</weight>
163.                <bold>true</bold>
164.               </font>
165.              </property>
166.             </item>
167.             <item row="0" column="2">
168.              <property name="text">
169.               <string>Reported By</string>
170.              </property>
171.              <property name="font">
172.               <font>
173.                <weight>75</weight>
```

```
174.                <bold>true</bold>
175.               </font>
176.              </property>
177.             </item>
178.            </widget>
179.           </item>
180.          </layout>
181.         </widget>
182.         <resources/>
183.         <connections/>
184.        </ui>
```

3.2.3 login.ui

```
1.  <?xml version="1.0" encoding="UTF-8"?>
2.  <ui version="4.0">
3.   <class>LoginDialog</class>
4.   <widget class="QDialog" name="LoginDialog">
5.    <property name="windowModality">
6.     <enum>Qt::ApplicationModal</enum>
7.    </property>
8.    <property name="enabled">
9.     <bool>true</bool>
10.   </property>
11.   <property name="geometry">
12.    <rect>
13.     <x>0</x>
14.     <y>0</y>
15.     <width>248</width>
16.     <height>145</height>
17.    </rect>
18.   </property>
19.   <property name="sizePolicy">
20.    <sizepolicy hsizetype="Fixed" vsizetype="Fixed">
21.     <horstretch>0</horstretch>
22.     <verstretch>0</verstretch>
23.    </sizepolicy>
24.   </property>
25.   <property name="minimumSize">
26.    <size>
27.     <width>248</width>
28.     <height>145</height>
29.    </size>
30.   </property>
31.   <property name="maximumSize">
32.    <size>
33.     <width>248</width>
34.     <height>147</height>
35.    </size>
36.   </property>
37.   <property name="windowTitle">
38.    <string>Photon - Login</string>
39.   </property>
40.   <property name="windowIcon">
41.    <iconset>
42.     <normaloff>icon.ico</normaloff>icon.ico</iconset>
43.   </property>
44.   <property name="sizeGripEnabled">
45.    <bool>false</bool>
46.   </property>
47.   <layout class="QFormLayout" name="formLayout">
48.    <item row="0" column="1">
49.     <widget class="QLineEdit" name="usernameInput">
50.      <property name="placeholderText">
```

```
51.       <string>Username</string>
52.      </property>
53.     </widget>
54.    </item>
55.    <item row="2" column="1">
56.     <widget class="QLineEdit" name="passwordInput">
57.      <property name="enabled">
58.       <bool>true</bool>
59.      </property>
60.      <property name="echoMode">
61.       <enum>QLineEdit::Password</enum>
62.      </property>
63.      <property name="placeholderText">
64.       <string>Password</string>
65.      </property>
66.     </widget>
67.    </item>
68.    <item row="3" column="1">
69.     <widget class="QPushButton" name="loginButton">
70.      <property name="text">
71.       <string>Login</string>
72.      </property>
73.     </widget>
74.    </item>
75.    <item row="4" column="0" colspan="2">
76.     <widget class="QLabel" name="errLabel">
77.      <property name="palette">
78.       <palette>
79.        <active>
80.         <colorrole role="WindowText">
81.          <brush brushstyle="SolidPattern">
82.           <color alpha="255">
83.            <red>255</red>
84.            <green>0</green>
85.            <blue>0</blue>
86.           </color>
87.          </brush>
88.         </colorrole>
89.        </active>
90.        <inactive>
91.         <colorrole role="WindowText">
92.          <brush brushstyle="SolidPattern">
93.           <color alpha="255">
94.            <red>255</red>
95.            <green>0</green>
96.            <blue>0</blue>
97.           </color>
98.          </brush>
99.         </colorrole>
100.             </inactive>
101.             <disabled>
102.              <colorrole role="WindowText">
103.               <brush brushstyle="SolidPattern">
104.                <color alpha="255">
105.                 <red>120</red>
106.                 <green>120</green>
107.                 <blue>120</blue>
108.                </color>
109.               </brush>
110.              </colorrole>
111.             </disabled>
112.            </palette>
113.           </property>
114.           <property name="text">
115.            <string/>
```

```
            </property>
           </widget>
          </item>
          <item row="5" column="0" colspan="2">
           <widget class="QPushButton" name="newAccButton">
            <property name="text">
             <string>Create New Account</string>
            </property>
           </widget>
          </item>
         </layout>
        </widget>
        <resources/>
        <connections/>
       </ui>
```

3.2.4 mainWindow.ui

```
<?xml version="1.0" encoding="UTF-8"?>
<ui version="4.0">
 <class>MainWindow</class>
 <widget class="QMainWindow" name="MainWindow">
  <property name="geometry">
   <rect>
    <x>0</x>
    <y>0</y>
    <width>673</width>
    <height>426</height>
   </rect>
  </property>
  <property name="palette">
   <palette>
    <active>
     <colorrole role="Shadow">
      <brush brushstyle="SolidPattern">
       <color alpha="255">
        <red>170</red>
        <green>255</green>
        <blue>127</blue>
       </color>
      </brush>
     </colorrole>
    </active>
    <inactive>
     <colorrole role="Shadow">
      <brush brushstyle="SolidPattern">
       <color alpha="255">
        <red>170</red>
        <green>255</green>
        <blue>127</blue>
       </color>
      </brush>
     </colorrole>
    </inactive>
    <disabled>
     <colorrole role="Shadow">
      <brush brushstyle="SolidPattern">
       <color alpha="255">
        <red>170</red>
        <green>255</green>
        <blue>127</blue>
       </color>
      </brush>
     </colorrole>
```

```
47.     </disabled>
48.    </palette>
49.   </property>
50.   <property name="windowTitle">
51.    <string>Photon - IM Client</string>
52.   </property>
53.   <property name="windowIcon">
54.    <iconset>
55.     <normaloff>icon.ico</normaloff>icon.ico</iconset>
56.   </property>
57.   <property name="layoutDirection">
58.    <enum>Qt::LeftToRight</enum>
59.   </property>
60.   <property name="iconSize">
61.    <size>
62.     <width>32</width>
63.     <height>32</height>
64.    </size>
65.   </property>
66.   <widget class="QWidget" name="centralwidget">
67.    <layout class="QGridLayout" name="gridLayout">
68.     <property name="leftMargin">
69.      <number>10</number>
70.     </property>
71.     <property name="topMargin">
72.      <number>10</number>
73.     </property>
74.     <property name="rightMargin">
75.      <number>10</number>
76.     </property>
77.     <property name="bottomMargin">
78.      <number>9</number>
79.     </property>
80.     <item row="1" column="2">
81.      <widget class="QLabel" name="userCountLabel">
82.       <property name="sizePolicy">
83.        <sizepolicy hsizetype="Preferred" vsizetype="Preferred">
84.         <horstretch>0</horstretch>
85.         <verstretch>0</verstretch>
86.        </sizepolicy>
87.       </property>
88.       <property name="layoutDirection">
89.        <enum>Qt::LeftToRight</enum>
90.       </property>
91.       <property name="text">
92.        <string>Users Online: 0</string>
93.       </property>
94.       <property name="alignment">
95.        <set>Qt::AlignLeading|Qt::AlignLeft|Qt::AlignVCenter</set>
96.       </property>
97.      </widget>
98.     </item>
99.     <item row="1" column="0">
100.           <widget class="QLabel" name="label">
101.            <property name="text">
102.             <string>Using Photon, a Python IM client by Sam Poirier</strin
    g>
103.            </property>
104.           </widget>
105.          </item>
106.          <item row="5" column="1">
107.           <widget class="QPushButton" name="messageInputButton">
108.            <property name="cursor">
109.             <cursorShape>PointingHandCursor</cursorShape>
110.            </property>
```

```
111.              <property name="text">
112.               <string>Send</string>
113.              </property>
114.              <property name="iconSize">
115.               <size>
116.                <width>20</width>
117.                <height>20</height>
118.               </size>
119.              </property>
120.             </widget>
121.            </item>
122.            <item row="5" column="0">
123.             <widget class="QLineEdit" name="messageInput">
124.              <property name="placeholderText">
125.               <string>Message</string>
126.              </property>
127.             </widget>
128.            </item>
129.            <item row="1" column="1">
130.             <widget class="QLabel" name="usernameLabel">
131.              <property name="sizePolicy">
132.               <sizepolicy hsizetype="Preferred" vsizetype="Preferred">
133.                <horstretch>0</horstretch>
134.                <verstretch>0</verstretch>
135.               </sizepolicy>
136.              </property>
137.              <property name="layoutDirection">
138.               <enum>Qt::LeftToRight</enum>
139.              </property>
140.              <property name="text">
141.               <string>Not Logged in</string>
142.              </property>
143.              <property name="alignment">
144.               <set>Qt::AlignRight|Qt::AlignTrailing|Qt::AlignVCenter</set>
145.              </property>
146.             </widget>
147.            </item>
148.            <item row="4" column="2">
149.             <widget class="QTextEdit" name="userListBox">
150.              <property name="enabled">
151.               <bool>true</bool>
152.              </property>
153.              <property name="maximumSize">
154.               <size>
155.                <width>80</width>
156.                <height>16777215</height>
157.               </size>
158.              </property>
159.              <property name="palette">
160.               <palette>
161.                <active>
162.                 <colorrole role="Base">
163.                  <brush brushstyle="SolidPattern">
164.                   <color alpha="255">
165.                    <red>240</red>
166.                    <green>240</green>
167.                    <blue>240</blue>
168.                   </color>
169.                  </brush>
170.                 </colorrole>
171.                </active>
172.                <inactive>
173.                 <colorrole role="Base">
174.                  <brush brushstyle="SolidPattern">
175.                   <color alpha="255">
```

```
                    <red>240</red>
                    <green>240</green>
                    <blue>240</blue>
                   </color>
                  </brush>
                 </colorrole>
                </inactive>
                <disabled>
                 <colorrole role="Base">
                  <brush brushstyle="SolidPattern">
                   <color alpha="255">
                    <red>240</red>
                    <green>240</green>
                    <blue>240</blue>
                   </color>
                  </brush>
                 </colorrole>
                </disabled>
               </palette>
              </property>
              <property name="readOnly">
               <bool>true</bool>
              </property>
             </widget>
            </item>
            <item row="4" column="0" colspan="2">
             <widget class="QScrollArea" name="messageScrollArea">
              <property name="palette">
               <palette>
                <active>
                 <colorrole role="Base">
                  <brush brushstyle="SolidPattern">
                   <color alpha="255">
                    <red>255</red>
                    <green>255</green>
                    <blue>255</blue>
                   </color>
                  </brush>
                 </colorrole>
                 <colorrole role="Window">
                  <brush brushstyle="SolidPattern">
                   <color alpha="255">
                    <red>255</red>
                    <green>255</green>
                    <blue>255</blue>
                   </color>
                  </brush>
                 </colorrole>
                </active>
                <inactive>
                 <colorrole role="Base">
                  <brush brushstyle="SolidPattern">
                   <color alpha="255">
                    <red>255</red>
                    <green>255</green>
                    <blue>255</blue>
                   </color>
                  </brush>
                 </colorrole>
                 <colorrole role="Window">
                  <brush brushstyle="SolidPattern">
                   <color alpha="255">
                    <red>255</red>
                    <green>255</green>
                    <blue>255</blue>
```

```
                    </color>
                   </brush>
                  </colorrole>
                 </inactive>
                 <disabled>
                  <colorrole role="Base">
                   <brush brushstyle="SolidPattern">
                    <color alpha="255">
                     <red>255</red>
                     <green>255</green>
                     <blue>255</blue>
                    </color>
                   </brush>
                  </colorrole>
                  <colorrole role="Window">
                   <brush brushstyle="SolidPattern">
                    <color alpha="255">
                     <red>255</red>
                     <green>255</green>
                     <blue>255</blue>
                    </color>
                   </brush>
                  </colorrole>
                 </disabled>
                </palette>
               </property>
               <property name="autoFillBackground">
                <bool>true</bool>
               </property>
               <property name="horizontalScrollBarPolicy">
                <enum>Qt::ScrollBarAlwaysOff</enum>
               </property>
               <property name="widgetResizable">
                <bool>true</bool>
               </property>
               <widget class="QWidget" name="scrollAreaWidgetContents">
                <property name="geometry">
                 <rect>
                  <x>0</x>
                  <y>0</y>
                  <width>565</width>
                  <height>308</height>
                 </rect>
                </property>
                <layout class="QGridLayout" name="gridLayout_2">
                 <property name="leftMargin">
                  <number>0</number>
                 </property>
                 <property name="topMargin">
                  <number>0</number>
                 </property>
                 <property name="rightMargin">
                  <number>0</number>
                 </property>
                 <property name="bottomMargin">
                  <number>0</number>
                 </property>
                 <item row="0" column="0">
                  <widget class="QWidget" name="messageWidget" native="true">

                   <property name="sizePolicy">
                    <sizepolicy hsizetype="Preferred" vsizetype="Expanding">
                     <horstretch>0</horstretch>
                     <verstretch>0</verstretch>
                    </sizepolicy>
```

```
                </property>
                <property name="autoFillBackground">
                 <bool>false</bool>
                </property>
                <widget class="QWidget" name="formLayoutWidget">
                 <property name="geometry">
                  <rect>
                   <x>1</x>
                   <y>4</y>
                   <width>562</width>
                   <height>250</height>
                  </rect>
                 </property>
                 <layout class="QFormLayout" name="messageLayout">
                  <property name="sizeConstraint">
                   <enum>QLayout::SetNoConstraint</enum>
                  </property>
                  <property name="fieldGrowthPolicy">
                   <enum>QFormLayout::FieldsStayAtSizeHint</enum>
                  </property>
                  <property name="rowWrapPolicy">
                   <enum>QFormLayout::WrapLongRows</enum>
                  </property>
                  <property name="horizontalSpacing">
                   <number>0</number>
                  </property>
                  <property name="verticalSpacing">
                   <number>0</number>
                  </property>
                 </layout>
                </widget>
               </widget>
              </item>
             </layout>
            </widget>
           </widget>
          </item>
          <item row="0" column="2">
           <widget class="QPushButton" name="adminSettingsButton">
            <property name="text">
             <string>Admin Settings</string>
            </property>
           </widget>
          </item>
         </layout>
        </widget>
        <widget class="QStatusBar" name="statusbar"/>
       </widget>
       <resources/>
       <connections/>
      </ui>
```

3.2.5 message.ui

```
<?xml version="1.0" encoding="UTF-8"?>
<ui version="4.0">
 <class>Form</class>
 <widget class="QWidget" name="Form">
  <property name="geometry">
   <rect>
    <x>0</x>
    <y>0</y>
    <width>420</width>
    <height>23</height>
```

```
   </rect>
  </property>
  <property name="sizePolicy">
   <sizepolicy hsizetype="Expanding" vsizetype="Expanding">
    <horstretch>0</horstretch>
    <verstretch>0</verstretch>
   </sizepolicy>
  </property>
  <property name="minimumSize">
   <size>
    <width>0</width>
    <height>0</height>
   </size>
  </property>
  <property name="maximumSize">
   <size>
    <width>16777215</width>
    <height>16777215</height>
   </size>
  </property>
  <property name="windowTitle">
   <string>Form</string>
  </property>
  <property name="autoFillBackground">
   <bool>true</bool>
  </property>
  <layout class="QGridLayout" name="gridLayout">
   <property name="leftMargin">
    <number>4</number>
   </property>
   <property name="topMargin">
    <number>2</number>
   </property>
   <property name="rightMargin">
    <number>4</number>
   </property>
   <property name="bottomMargin">
    <number>0</number>
   </property>
   <property name="horizontalSpacing">
    <number>2</number>
   </property>
   <item row="0" column="6">
    <widget class="QPushButton" name="messageOptionBtn">
     <property name="styleSheet">
      <string notr="true">background-color:transparent;
border:none</string>
     </property>
     <property name="text">
      <string/>
     </property>
     <property name="icon">
      <iconset>
       <normaloff>verticalEllipsis.png</normaloff>verticalEllipsis.png</icon
set>
     </property>
    </widget>
   </item>
   <item row="0" column="1">
    <widget class="QLabel" name="timeLabel">
     <property name="palette">
      <palette>
       <active>
        <colorrole role="WindowText">
         <brush brushstyle="SolidPattern">
```

```
          <color alpha="255">
           <red>99</red>
           <green>99</green>
           <blue>99</blue>
          </color>
         </brush>
        </colorrole>
       </active>
       <inactive>
        <colorrole role="WindowText">
         <brush brushstyle="SolidPattern">
          <color alpha="255">
           <red>99</red>
           <green>99</green>
           <blue>99</blue>
          </color>
         </brush>
        </colorrole>
       </inactive>
       <disabled>
        <colorrole role="WindowText">
         <brush brushstyle="SolidPattern">
          <color alpha="255">
           <red>120</red>
           <green>120</green>
                  <blue>120</blue>
                 </color>
                </brush>
               </colorrole>
              </disabled>
             </palette>
            </property>
            <property name="layoutDirection">
             <enum>Qt::LeftToRight</enum>
            </property>
            <property name="text">
             <string>timestamp</string>
            </property>
            <property name="alignment">
             <set>Qt::AlignLeading|Qt::AlignLeft|Qt::AlignVCenter</set>
            </property>
           </widget>
          </item>
          <item row="0" column="2">
           <widget class="QLabel" name="usernameLabel">
            <property name="palette">
             <palette>
              <active>
               <colorrole role="Shadow">
                <brush brushstyle="SolidPattern">
                 <color alpha="255">
                  <red>240</red>
                  <green>240</green>
                  <blue>240</blue>
                 </color>
                </brush>
               </colorrole>
              </active>
              <inactive>
               <colorrole role="Shadow">
                <brush brushstyle="SolidPattern">
                 <color alpha="255">
                  <red>240</red>
                  <green>240</green>
                  <blue>240</blue>
```

```
                </color>
               </brush>
              </colorrole>
             </inactive>
             <disabled>
              <colorrole role="Shadow">
               <brush brushstyle="SolidPattern">
                <color alpha="255">
                 <red>240</red>
                 <green>240</green>
                 <blue>240</blue>
                </color>
               </brush>
              </colorrole>
             </disabled>
            </palette>
           </property>
           <property name="font">
            <font>
             <pointsize>8</pointsize>
             <weight>75</weight>
             <bold>true</bold>
            </font>
           </property>
           <property name="text">
            <string>username</string>
           </property>
          </widget>
         </item>
         <item row="0" column="3">
          <widget class="QLabel" name="messageLabel">
           <property name="sizePolicy">
            <sizepolicy hsizetype="Expanding" vsizetype="Fixed">
             <horstretch>0</horstretch>
             <verstretch>0</verstretch>
            </sizepolicy>
           </property>
           <property name="maximumSize">
            <size>
             <width>16777215</width>
             <height>14</height>
            </size>
           </property>
           <property name="text">
            <string>message</string>
           </property>
           <property name="textInteractionFlags">
            <set>Qt::LinksAccessibleByMouse|Qt::TextSelectableByKeyboard|Qt
    ::TextSelectableByMouse</set>
           </property>
          </widget>
         </item>
        </layout>
       </widget>
       <resources/>
       <connections/>
      </ui>
```

3.2.6 messageOptions.ui

```
1.  <?xml version="1.0" encoding="UTF-8"?>
2.  <ui version="4.0">
3.   <class>Dialog</class>
4.   <widget class="QDialog" name="Dialog">
```

```
5.    <property name="geometry">
6.     <rect>
7.      <x>0</x>
8.      <y>0</y>
9.      <width>600</width>
10.     <height>72</height>
11.    </rect>
12.   </property>
13.   <property name="minimumSize">
14.    <size>
15.     <width>329</width>
16.     <height>72</height>
17.    </size>
18.   </property>
19.   <property name="maximumSize">
20.    <size>
21.     <width>16777215</width>
22.     <height>72</height>
23.    </size>
24.   </property>
25.   <property name="windowTitle">
26.    <string>Message Options</string>
27.   </property>
28.   <property name="modal">
29.    <bool>true</bool>
30.   </property>
31.   <layout class="QGridLayout" name="gridLayout_2">
32.    <item row="0" column="0">
33.     <layout class="QGridLayout" name="gridLayout">
34.      <item row="1" column="0">
35.       <widget class="QLabel" name="timeLabel">
36.        <property name="palette">
37.         <palette>
38.          <active>
39.           <colorrole role="WindowText">
40.            <brush brushstyle="SolidPattern">
41.             <color alpha="255">
42.              <red>99</red>
43.              <green>99</green>
44.              <blue>99</blue>
45.             </color>
46.            </brush>
47.           </colorrole>
48.          </active>
49.          <inactive>
50.           <colorrole role="WindowText">
51.            <brush brushstyle="SolidPattern">
52.             <color alpha="255">
53.              <red>99</red>
54.              <green>99</green>
55.              <blue>99</blue>
56.             </color>
57.            </brush>
58.           </colorrole>
59.          </inactive>
60.          <disabled>
61.           <colorrole role="WindowText">
62.            <brush brushstyle="SolidPattern">
63.             <color alpha="255">
64.              <red>120</red>
65.              <green>120</green>
66.              <blue>120</blue>
67.             </color>
68.            </brush>
69.           </colorrole>
```

```
70.           </disabled>
71.          </palette>
72.         </property>
73.         <property name="text">
74.          <string>0:00:00</string>
75.         </property>
76.        </widget>
77.       </item>
78.       <item row="4" column="0" colspan="4">
79.        <widget class="QLineEdit" name="reportReason">
80.         <property name="inputMask">
81.          <string/>
82.         </property>
83.         <property name="text">
84.          <string/>
85.         </property>
86.         <property name="placeholderText">
87.          <string>Report Reason</string>
88.         </property>
89.        </widget>
90.       </item>
91.       <item row="1" column="4">
92.        <widget class="QPushButton" name="deleteButton">
93.         <property name="text">
94.          <string>Delete</string>
95.         </property>
96.        </widget>
97.       </item>
98.       <item row="1" column="3">
99.        <widget class="QPushButton" name="editButton">
100.             <property name="text">
101.              <string>Edit</string>
102.             </property>
103.            </widget>
104.           </item>
105.           <item row="1" column="2">
106.            <widget class="QLineEdit" name="editMessage">
107.             <property name="placeholderText">
108.              <string>Edited Text</string>
109.             </property>
110.            </widget>
111.           </item>
112.           <item row="4" column="4">
113.            <widget class="QPushButton" name="reportButton">
114.             <property name="text">
115.              <string>Report</string>
116.             </property>
117.            </widget>
118.           </item>
119.           <item row="1" column="1">
120.            <widget class="QLabel" name="usernameLabel">
121.             <property name="text">
122.              <string><html><head/><body><p><span style=" font-
    weight:600;">&lt;user&gt;</span></p></body></html></string>
123.             </property>
124.            </widget>
125.           </item>
126.          </layout>
127.         </item>
128.        </layout>
129.       </widget>
130.       <resources/>
131.       <connections/>
132.      </ui>
```

3.2.7 register.ui

```
<?xml version="1.0" encoding="UTF-8"?>
<ui version="4.0">
 <class>RegisterDialog</class>
 <widget class="QDialog" name="RegisterDialog">
  <property name="windowModality">
   <enum>Qt::ApplicationModal</enum>
  </property>
  <property name="enabled">
   <bool>true</bool>
  </property>
  <property name="geometry">
   <rect>
    <x>0</x>
    <y>0</y>
    <width>236</width>
    <height>156</height>
   </rect>
  </property>
  <property name="sizePolicy">
   <sizepolicy hsizetype="Fixed" vsizetype="Fixed">
    <horstretch>0</horstretch>
    <verstretch>0</verstretch>
   </sizepolicy>
  </property>
  <property name="minimumSize">
   <size>
    <width>236</width>
    <height>156</height>
   </size>
  </property>
  <property name="maximumSize">
   <size>
    <width>236</width>
    <height>156</height>
   </size>
  </property>
  <property name="windowTitle">
   <string>Photon - Register</string>
  </property>
  <property name="windowIcon">
   <iconset>
    <normaloff>icon.ico</normaloff>icon.ico</iconset>
  </property>
  <property name="sizeGripEnabled">
   <bool>false</bool>
  </property>
  <layout class="QFormLayout" name="formLayout">
   <item row="2" column="1">
    <widget class="QLineEdit" name="passwordInput1">
     <property name="enabled">
      <bool>true</bool>
     </property>
     <property name="echoMode">
      <enum>QLineEdit::Password</enum>
     </property>
     <property name="placeholderText">
      <string>Password</string>
     </property>
    </widget>
   </item>
   <item row="3" column="1">
    <widget class="QLineEdit" name="passwordInput2">
```

```
63.      <property name="echoMode">
64.       <enum>QLineEdit::Password</enum>
65.      </property>
66.      <property name="placeholderText">
67.       <string>Re-enter Password</string>
68.      </property>
69.     </widget>
70.    </item>
71.    <item row="6" column="1">
72.     <widget class="QPushButton" name="registerButton">
73.      <property name="text">
74.       <string>Register</string>
75.      </property>
76.     </widget>
77.    </item>
78.    <item row="7" column="0" colspan="2">
79.     <widget class="QLabel" name="errLabel">
80.      <property name="maximumSize">
81.       <size>
82.        <width>285</width>
83.        <height>26</height>
84.       </size>
85.      </property>
86.      <property name="palette">
87.       <palette>
88.        <active>
89.         <colorrole role="WindowText">
90.          <brush brushstyle="SolidPattern">
91.           <color alpha="255">
92.            <red>255</red>
93.            <green>0</green>
94.            <blue>0</blue>
95.           </color>
96.          </brush>
97.         </colorrole>
98.        </active>
99.        <inactive>
100.                <colorrole role="WindowText">
101.                 <brush brushstyle="SolidPattern">
102.                  <color alpha="255">
103.                   <red>255</red>
104.                   <green>0</green>
105.                   <blue>0</blue>
106.                  </color>
107.                 </brush>
108.                </colorrole>
109.               </inactive>
110.               <disabled>
111.                <colorrole role="WindowText">
112.                 <brush brushstyle="SolidPattern">
113.                  <color alpha="255">
114.                   <red>120</red>
115.                   <green>120</green>
116.                   <blue>120</blue>
117.                  </color>
118.                 </brush>
119.                </colorrole>
120.               </disabled>
121.              </palette>
122.             </property>
123.             <property name="text">
124.              <string/>
125.             </property>
126.            </widget>
127.           </item>
```

```
          <item row="0" column="1">
           <widget class="QLineEdit" name="usernameInput">
            <property name="placeholderText">
             <string>Username</string>
            </property>
           </widget>
          </item>
         </layout>
        </widget>
        <resources/>
        <connections/>
       </ui>
```

3.2.8 config.json

```
{
    "maxTransmissionSize": 40960,
    "debug": true,
    "commandChar": "/",
    "port": 9998
}
```

3.3 Libs

3.3.1 configManager.py

```
import json
import os.path

class ConfigManager():
    def __init__(self, file, defaultData):
        self.data = self.loadJson(file)
        if self.data == None:
            self.initJson(file, defaultData)
            self.data = self.loadJson(file)

    def loadJson(self, file):
        if os.path.isfile(file):
            with open(file, "r") as jsonFile:
                return json.load(jsonFile)
        else:
            return None

    def initJson(self, file, defaultData):
        jsonString = json.dumps(defaultData)
        with open(file, "w") as jsonFile:
            jsonFile.write(jsonString)


class ServerConfig(ConfigManager):
    def __init__(self, file):
        self.defaultData = {

            "dbFile": "photon.db",
            "infoLoggingEnabled": True,
            "maxTransmissionSize": 40960,
            "port": 9998

            }

        super().__init__(file, self.defaultData)

```

```
38. class ClientConfig(ConfigManager):
39.     def __init__(self, file):
40.         self.defaultData = {
41.
42.             "maxTransmissionSize": 40960,
43.             "debug": True,
44.             "commandChar": "/",
45.             "port": 9998
46.
47.             }
48.
49.         super().__init__(file, self.defaultData)
```

3.3.1 packets.py

```
1.  class Packet:
2.    """
3.    Base packet class. Can be used to signal something without the need to tra
    nsfer dataself.
4.
5.    Args:
6.      packetType (string): An identifier to determine what the packet denotes.
     Should be all caps.
7.    """
8.    def __init__(self, packetType):
9.      self.type = packetType
10.
11. class LoginRequestPacket(Packet):
12.   """
13.   Sends login details to the server.
14.
15.   Args:
16.     username (string): The username to login with.
17.     password (string): The password to login with. Should be hashed.
18.   """
19.   def __init__(self, username, password):
20.     Packet.__init__(self, "LOGINREQUEST")
21.     self.username = username
22.     self.password = password
23.
24. class LoginResponsePacket(Packet):
25.   """
26.   Sends login response to client.
27.
28.   Args:
29.     valid (bool): Whether the login was successful.
30.     err (string, optional): Why the login was not successful.
31.     id (int, optional): The user id of the account, if successfully logged i
    n.
32.     admin (bool, optional): Whether the user account is an admin.
33.   """
34.   def __init__(self, valid, userId="", err="", admin=False):
35.     Packet.__init__(self, "LOGINRESPONSE")
36.     self.valid = valid
37.     self.err = err
38.     self.id = userId
39.     self.admin = admin
40.
41. class RegisterPacket(Packet):
42.   """
43.   Asks the server to create a user account.
44.   Args:
45.     username (string): The username of the account to create.
```

```
46.     password (string): The password of the account to create. Should be hash
    ed.
47.   """
48.   def __init__(self, username, password):
49.     Packet.__init__(self, "CREATEUSER")
50.     self.username = username
51.     self.password = password
52.
53. class RegisterResponsePacket(Packet):
54.   """
55.   Informs the client if the account was created succesfully.
56.
57.   Args:
58.     valid (bool): Whether the account creation was a success.
59.     err (string, optional): The reason why the account creation was not a su
    ccess.
60.   """
61.   def __init__(self, valid, err=""):
62.     Packet.__init__(self, "REGISTERRESPONSE")
63.     self.valid = valid
64.     self.err = err
65.
66. class MessagePacket(Packet):
67.   """
68.   Sends a message to the client or the server.
69.
70.   Args:
71.     message (photonUtilities.Message): The message to send.
72.   """
73.   def __init__(self, message):
74.     Packet.__init__(self, "MESSAGE")
75.     self.message = message # Utilises Message class
76.
77. class MessageListPacket(Packet):
78.   """
79.   Sends multiple messages to the client or server.
80.
81.   Args:
82.     messageList (list of photonUtilities.Message): The list of messages to s
    end.
83.   """
84.   def __init__(self, messageList):
85.     Packet.__init__(self, "MESSAGELIST")
86.     self.messageList = messageList
87.
88. class OnlineUsersPacket(Packet):
89.   """
90.   Sends a list of the usernames of all clients which are online.
91.
92.   Args:
93.     userList (list of string): List of usernames of online clients.
94.   """
95.   def __init__(self, userList):
96.     Packet.__init__(self, "ONLINEUSERS")
97.     self.userList = userList
98.
99. class UserListPacket(Packet):
100.         """
101.         Sends a list of all users registered.
102.
103.         Args:
104.           userList (list of (int, string, bool)): A list of tuples conainin
    g the userId, username and whether they're admin.
105.         """
106.         def __init__(self, userList):
```

```
107.            Packet.__init__(self, "USERLIST")
108.            self.userList = userList
109.
110.        class RequestUserInfoPacket(Packet):
111.          """
112.          Requests details about a specific user.
113.
114.          Args:
115.            user (string): The username of the user to get the info of.
116.          """
117.          def __init__(self, user):
118.            Packet.__init__(self, "REQUESTUSERINFO")
119.            self.user = user
120.
121.        class UserInfoPacket(Packet):
122.          """
123.          Sends information about a specific user.
124.
125.          Args:
126.            id (int): The id of the user
127.            messageCount (int): The number of messages sent by the user.
128.            admin (bool): Whether the user is an admin.
129.            flags (list of (string, string, string, int): Contains info about
     reports for that user, corresponding to list of (reported message, report r
    eason, reporter name, reporter id).
130.          """
131.          def __init__(self, id, messageCount, admin, flags):
132.            Packet.__init__(self, "USERINFO")
133.            self.id = id
134.            self.messageCount = messageCount
135.            self.admin = admin
136.            self.flags = flags
137.
138.        class CommandPacket(Packet):
139.          """
140.          Contains information about an executed command
141.
142.          Args:
143.            command (string): The command that was executed.
144.            args (list of string): A list of the arguments supplied to the co
    mmand.
145.          """
146.          def __init__(self, command, args=[]):
147.            Packet.__init__(self, "COMMAND")
148.            self.command = command
149.            self.args = args
150.
151.        class CommandResponsePacket(Packet):
152.          """
153.          Contains information about the execution of a command.
154.
155.          Args:
156.            command (string): The command that was executed.
157.            success (bool): Whether the command was executed successfully.
158.            err (string, optional): The reason why the command could not exec
    ute.
159.            response (string, optional): The response of the command.
160.            timeSent (string, optional): The time that the command was execut
    ed.
161.          """
162.          def __init__(self, command, success, err="", response="", timeSent=
    ""):
163.            Packet.__init__(self, "COMMANDRESPONSE")
164.            self.command = command
165.            self.success = success
```

```
166.            self.err = err
167.            self.response = response
168.            self.timeSent = timeSent
169.
170.        class ReportPacket(Packet):
171.          """
172.          Tells the server to flag a message.
173.
174.          Args:
175.            messageId (int): The id of the reported message.
176.            reporterId (int): The id of the user who reported the message.
177.            reportReason (string): The reason that the message was flagged.
178.          """
179.          def __init__(self, messageId, reporterId, reportReason):
180.            Packet.__init__(self, "REPORTPACKET")
181.            self.messageId = messageId
182.            self.reporterId = reporterId
183.            self.reportReason = reportReason
184.
185.        class DeleteMessagePacket(Packet):
186.          def __init__(self, messageId):
187.            Packet.__init__(self, "DELETEMESSAGE")
188.            self.messageId = messageId
189.
190.        class EditMessagePacket(Packet):
191.          def __init__(self, messageId, newContents):
192.            Packet.__init__(self, "EDITMESSAGE")
193.            self.messageId = messageId
194.            self.newContents = newContents
195.
196.        class SetAdminStatusPacket(Packet):
197.          def __init__(self, admin, userId):
198.            Packet.__init__(self, "SETADMINSTATUS")
199.            self.userId = userId
200.            self.admin = admin
```

3.3.1 photonUtilities.py

```
1.  import pickle
2.  import traceback
3.  import datetime
4.
5.  # Global Colours
6.  BLACK = "#000000"
7.  COMMANDERROR = "#ff3030"
8.  INFO = "#636363"
9.
10.
11. class CircularQueue():
12.   """
13.   A custom circular queue implementation.
14.   """
15.   def __init__(self, maxSize):
16.     if maxSize < 1:
17.       raise ValueError("Queue size must be at least 1")
18.     self.data = [''] * maxSize
19.     self.rear = -1
20.     self.front= 0
21.     self.size = 0
22.     self.maxSize = maxSize
23.
24.   def enQueue(self, item):
25.     """
26.     Add an item to the queue.
```

```
27.
28.     Args:
29.       item (*): Item to add to the queue.
30.     """
31.     if self.size == self.maxSize:
32.       raise ValueError("Cannot enqueue when the queue is full")
33.     else:
34.       self.rear = (self.rear + 1) % self.maxSize
35.       self.data[self.rear] = item
36.       self.size = self.size + 1
37.
38.   def deQueue(self):
39.     """
40.     Removes an item from the queue.
41.
42.     Returns:
43.       (*): The item at the front of the queue.
44.     """
45.     self.front += 1
46.     self.size -= 1
47.     return self.data[self.front-1]
48.
49.   def isFull(self):
50.     """ Determines if the queue is full. """
51.     return self.size == self.rear
52.
53.   def isEmpty(self):
54.     """ Determines if the queue is empty. """
55.     return self.size == 0
56.
57.
58. class Message():
59.   """
60.   Represents a message.
61.
62.   Args:
63.     senderId (int): The user id of the message sender.
64.     senderName (string, optional): The username of the
65.     contents (string, optional): The actual message content
66.     timeSent (string, optional): The time that the message was sent.
67.     recipientId (int, optional): The user id of the user that sent the messa
    ge.
68.     colour (string, optional): The colour to display the message as.
69.     messageId (int, optional): The id of the message.
70.   """
71.   def __init__(self, senderId="", senderName="", contents="", timeSent="", r
    ecipientId=1, colour="#000000", messageId=""):
72.     self.senderId = senderId
73.     self.senderName = senderName
74.     self.contents = contents
75.     self.timeSent = timeSent
76.     self.recipientId = recipientId
77.     self.colour = colour
78.     self.messageId = messageId
79.
80.
81. def hashString(string):
82.   """
83.   Custom implementation of a simple one way hashing algorithm
84.
85.   Args:
86.     string (string): The string to hash.
87.
88.   Returns:
89.     (string): The hashed string.
```

```
    """
    bitValueChunk = ""
    bitSum = 0

    for char in string:
      bitSum += ord(char)
      bitValue = format(ord(char), 'b') # Convert char to binary
      bitValueChunk += bitValue # Apend to 'binary chunk'

    n = 9
          bitValues = [bitValueChunk[i:i+n] for i in range(0, len(bitValueChunk), n)] # Split 'binary chunk' into list of 9 bit binary numbers

          moddedBitChunk = ""
          for bitValue in bitValues:
            bitValue = int(bitValue)
            moddedBitValue = bitValue + (bitValue % 37) # Modulo is a one way function, so we modulo by a prime as the core of the hash. This is the step that ensures the hash is unidirectional
            moddedBitValue = moddedBitValue * bitSum # Multiply by sum of the ascii values of the chars to ensure similar input strings look different
            moddedBitChunk += format(moddedBitValue, 'b') # Convert into binary again

          n = 6 # 6 bit chunks to avoid strange characters
          moddedBitValues = [moddedBitChunk[i:i+n] for i in range(0, len(moddedBitChunk), n)] # Split modded 'binary chunk' into list of 8 bit binary numbers

          hashed = ""
          for moddedBitValue in moddedBitValues:
            hashed += chr(int(moddedBitValue, 2) + 33) # Convert binary value into decimal value then into the corresponding character, skipping the first 33 as they are non-printing/whitespace

          return hashed


        def integerMergeSort(mergelist):
          """
          Custom implementation of a mergesort algorithm for integers.

          Args:
            mergeList (list of int): The list to sort.

          Returns:
            (list of int): The sorted list.
          """
          if len(mergelist) > 1:
              mid = len(mergelist) // 2 # Perform integer division
              lefthalf = mergelist[:mid] # Left half of merglist into lefthalf
              righthalf = mergelist[mid:] # Right half of merglist into right half
              lefthalf = MergeSort(lefthalf)
              righthalf = MergeSort(righthalf)

              i = 0
              j = 0
              k = 0
              while i < len(lefthalf) and j < len(righthalf):
                  if lefthalf[i] < righthalf[j]:
                      mergelist[k] = lefthalf[i]
                      i += 1
                  else:
```

```
144.                     mergelist[k] = righthalf[j]
145.                     j += 1
146.                 k += 1
147.
148.             # Check if left half has elements not merged
149.             while i < len(lefthalf):
150.                 mergelist[k] = lefthalf[i] # If so, add to mergelist
151.                 i += 1
152.                 k += 1
153.             # Check if right half has elements not merged
154.             while j < len(righthalf):
155.                 mergelist[k] = righthalf[j] # If so, add to mergelist
156.                 j += 1
157.                 k += 1
158.         return mergelist
159.
160.
161.       def stringListMergeSort(mergelist):
162.         """
163.         Custom implementation of a mergesort algorithm for strings.
164.
165.         Args:
166.           mergeList (list of string): The list to sort.
167.
168.         Returns:
169.           (list of string): The sorted list.
170.         """
171.         if len(mergelist) > 1:
172.           mid = len(mergelist) // 2 # Perform integer division
173.           lefthalf = mergelist[:mid] # Left half of merglist into lefthalf

174.           righthalf = mergelist[mid:] # Right half of merglist into rightha
    lf
175.           lefthalf = stringListMergeSort(lefthalf)
176.           righthalf = stringListMergeSort(righthalf)
177.
178.           i = 0
179.           j = 0
180.           k = 0
181.           while i < len(lefthalf) and j < len(righthalf):
182.               l = 0
183.               while ord(lefthalf[i][l]) == ord(righthalf[j][l]) and l < len
    (lefthalf[i])-1 and l < len(righthalf[j])-
    1: # If the charachers are the same, we must look at the next one until the
    end
184.                   l += 1
185.               if ord(lefthalf[i][l]) < ord(righthalf[j][l]):
186.                   mergelist[k] = lefthalf[i]
187.                   i += 1
188.               else:
189.                   mergelist[k] = righthalf[j]
190.                   j += 1
191.               k += 1
192.
193.           # Check if left half has elements not merged
194.           while i < len(lefthalf):
195.               mergelist[k] = lefthalf[i] # If so, add to mergelist
196.               i += 1
197.               k += 1
198.           # Check if right half has elements not merged
199.           while j < len(righthalf):
200.               mergelist[k] = righthalf[j] # If so, add to mergelist
201.               j += 1
202.               k += 1
203.         return mergelist
```

```
204.
205.
206.       def reportError(exception="", logger=None):
207.         """ Easier name for traceback.print_exc(). """
208.         traceback.print_exc()
209.         if logger != None:
210.           logger.log(repr(exception))
211.
212.
213.       def getDateTime():
214.         """ Gets the current time in yy/mm/dd HH:MM format. """
215.         return datetime.datetime.now().strftime("%y-%m-%d %H:%M")
216.
217.
218.       def formatUsername(name):
219.         """
220.         Formats a string to confirm with username display style.
221.
222.         Args:
223.           name (string): The name to format.
224.
225.         Returns:
226.           (string): The formatted string.
227.         """
228.         if name == "" or name == "SERVER":
229.           return ""
230.         else:
231.           return "<" + name + ">: "
232.
233.
234.       def formatDateTime(time):
235.         """
236.         Formats a string to confirm with time display style.
237.
238.         Args:
239.           time (string): The time to format.
240.
241.         Returns:
242.           (string): The formatted string.
243.         """
244.         if time == "":
245.           return ""
246.         else:
247.           return "" + time + " | "
248.
249.
250.       def generateJoinLeaveMessage(direction, username):
251.         """
252.         Generates a standard message notifying of when users join/leave the
    server.
253.
254.         Args:
255.           direction (string): Contains whether the user 'joined' or 'left'
   the server.
256.           username (string): The user who joined or left.
257.
258.         Returns:
259.           (Message): The generated message.
260.         """
261.         return Message(1, "SERVER", "_" + username + " has " + direction +
    " the server_", getDateTime(), colour=INFO)
262.
263.
264.       def debugPrint(message, debug):
265.         """
```

```
266.         Conditional Print depending on whether launched in debug mode or no
    t.
267.
268.         Args:
269.           message (string): The message to print.
270.           debug (bool): Whether the message should be printed.
271.         """
272.         if debug:
273.           print(message)
274.
275.
276.       def encode(packet):
277.         """ Better name for pickle.dumps() """
278.         return pickle.dumps(packet)
279.       def decode(packet):
280.         """ Better name for pickle.loads() """
281.         return pickle.loads(packet)
```

4 Testing

4.1 Testing Strategy

The main method of testing that I plan to implement is Black Box Testing, essentially ensuring that specific inputs lead to the correct outputs. I will also be utilising end user testing as this is the best method to find bugs as the user has no knowledge about how the program is written, so uses it more naturally.

4.2 Test Tables

Test No.	1
Test Purpose	Can users create new valid and invalid accounts?
Test Description	When the program is started, determine if the user can successfully create a new account to login with.
Test Data	a) Username: “testUser”, Password: “testPassword” b) Username: “admin”, Password: “admin” c) Username: “”, Password: “password” d) Username: “newUser”, Password: “”
Expected Result	a) Successful creation of new account b) Unsuccessful creation of account, user already exists error appears c) Unsuccessful creation of account, username is empty error appears d) Unsuccessful creation of account, password is empty error appears
Actual Result	Pass – inline with expected results
Relavent Objectives	2a, 2b, 2c, 2d

Test No.	2
Test Purpose	Can users login to accounts?
Test Description	When the program is started, determine if the user can successfully login with their account.
Test Data	a) Username: “testUser”, Password: “testPassword” b) Username: “admin”, Password: “admin” c) Username: “admin”, Password: “testPassword” d) Username: “administrator”, Password: “admin”

Expected Result	a) Successful user login b) Successful user login c) Unsuccessful login, incorrect password error appears d) Unsuccessful login, unknown user error appears
Actual Result	a) Pass b) Pass c) Fail – no error message was displayed d) Fail – no error message was displayed
Relavent Objectives	1a, 1b, 1c, 1d

Test No.	3
Test Purpose	Do Logged in users appear in the online user list?
Test Description	Once a user has logged in, they should appear alphabetically in the list on the right hand side of the client window.
Test Data	a) One user connected, “testUser” b) “testUser” connected then “user” connected c) “testUser” connected, followed by “user” then “admin”
Expected Result	a) “testUser” appears in the logged in user box b) “testUser” appears in the logged in user box, with “user” underneath c) “testUser” appears in the logged in user box, with “user” underneath and “admin” at the top
Actual Result	Pass – inline with expected results
Relavent Objectives	3d

Test No.	4
Test Purpose	Can users send/receive messages?
Test Description	Once a user has logged in, send a message and check that it is received back from the server successfully.
Test Data	a) “Hello World” b) “” c) “ ’); DROP TABLE Message; ”
Expected Result	a) “Hello World” appears in the chat window and is removed from the input box b) No message is sent or received d) “ ’); DROP TABLE Message; ” appears in the chat window and is removed from the input box
Actual Result	Pass – inline with expected results
Relavent Objectives	3a, 3b, 3c

Test No.	5
Test Purpose	Are messages styled correctly?
Test Description	There is support for making messages bold, underlined, italicised or with a strikethrough using balsamiq markup
Test Data	a) “*Hello World*” b) “_Hello World_” c) “!Hello World!” d) “~Hello World~” e) “_*Hello world*_”

	f) "_~!*Hello World*!~_" g) "~Hello World" h) "Hello World!"
Expected Result	a) "Hello World" appears in the chat window and is bold b) "Hello World" appears in the chat window and is italicised c) "Hello World" appears in the chat window and is underlined f) "Hello World" appears in the chat window and is struck through g) "~Hello World" appears in the chat window f) "Hello World!" appears in the chat window
Actual Result	Pass – inline with expected results
Relavent Objectives	19

Test No.	6
Test Purpose	Can messages be edited?
Test Description	When clicking the kebab menu (⋮) on a message it should open a menu allowing you to edit your message, provided it was your message.
Test Data	a) Attempting to edit own message to "this message has been edited" b) Attempting to edit another user's message to "this message has been edited too"
Expected Result	a) Message is successfully edited b) Unable to edit message
Actual Result	Pass – inline with expected results
Relavent Objectives	12

Test No.	7
Test Purpose	Can messages be deleted
Test Description	When clicking the kebab menu (⋮) on a message it should open a menu allowing you to delete the message, provided it was your message or you are an admin.
Test Data	a) Attempting to delete own message as standard user b) Attempting to delete other's message as standard user c) Attempting to delete other's message as admin user
Expected Result	a) Message is successfully deleted b) Unable to delete message c) Message is successfully deleted
Actual Result	Pass – inline with expected results
Relavent Objectives	17

Test No.	8
Test Purpose	Can messages be reported?
Test Description	When clicking the kebab menu (⋮) on a message it should open a menu allowing you to report the message, provided it was not your message.
Test Data	a) Attempting to report own message b) Attempting to report other's message
Expected Result	a) Unable to report message b) Message is successfully reported
Actual Result	Pass – inline with expected results
Relavent Objectives	13

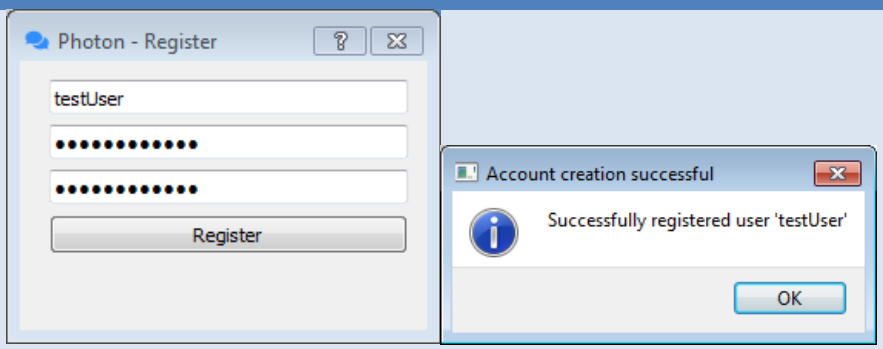
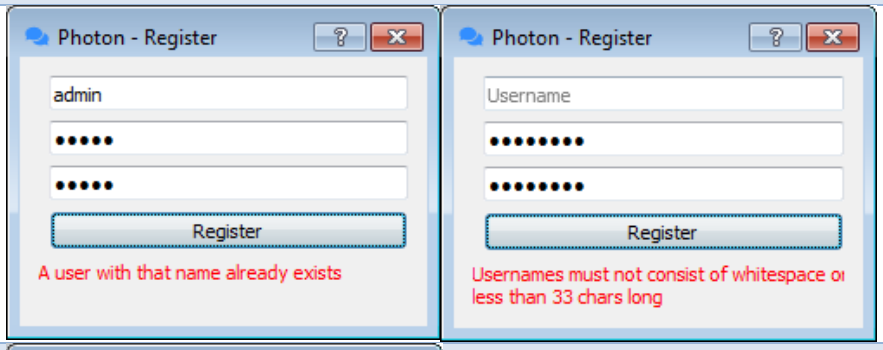
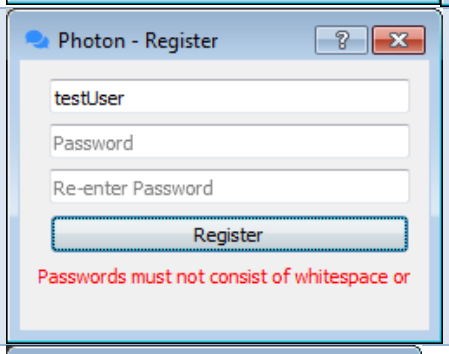
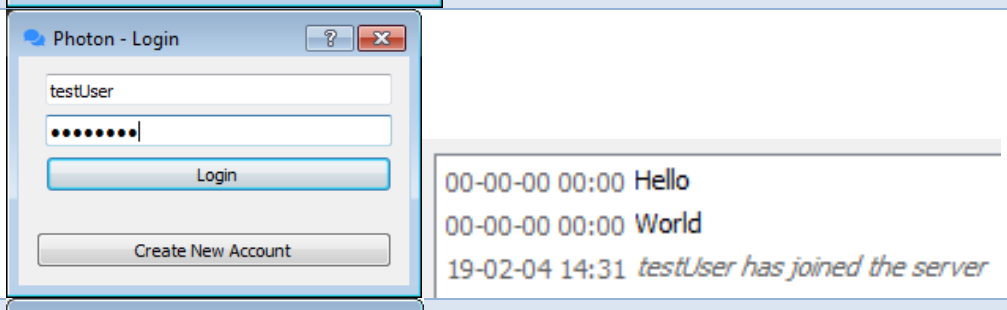
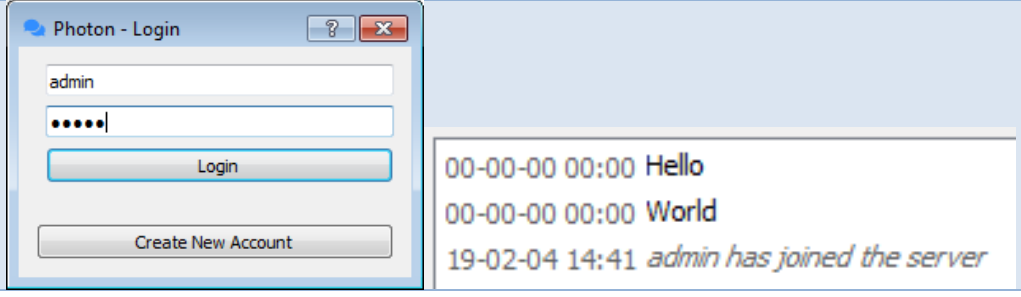
Test No.	9
Test Purpose	Can admins promote users?
Test Description	When clicking admin settings menu, they should be able to promote/demote users to admin.
Test Data	a) Attempting to promote standard user c) Attempting to demote admin
Expected Result	a) User is promoted, 'promote' button changes to demote b) User is demoted, 'demote' button changes to promote
Actual Result	Pass – inline with expected results
Relavent Objectives	16

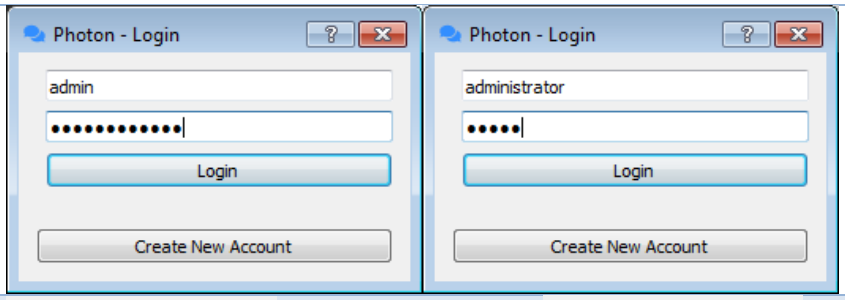
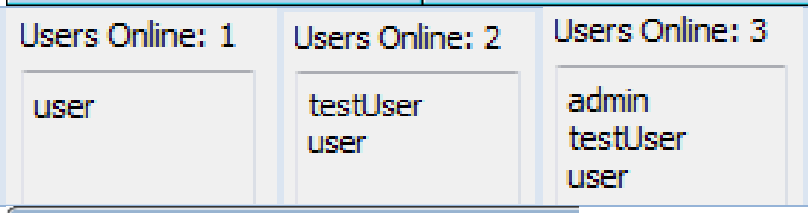
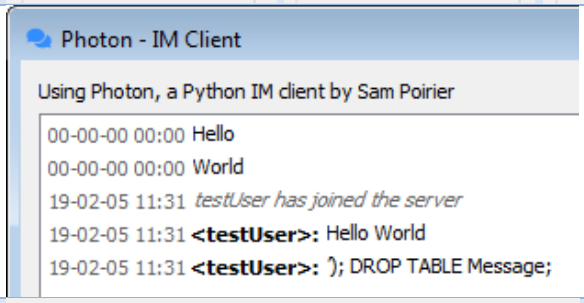
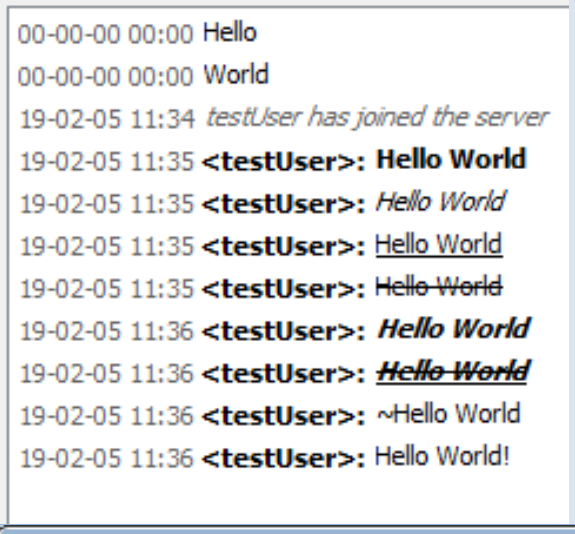
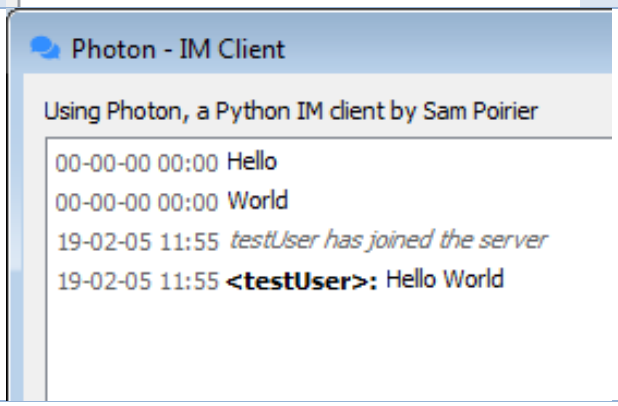
Test No.	10
Test Purpose	Can admins review user reports?
Test Description	When clicking admin settings menu, they should be view a list of reports for a user
Test Data	a) Viewing a user with 1 report registered against them c) Viewing a user with 5 reports registered against them d) Viewing a user with 0 reports registered against them
Expected Result	a) User info page displays report b) User info page displays 5 reports c) User info page displays 0 reports
Actual Result	Pass – inline with expected results
Relavent Objectives	14

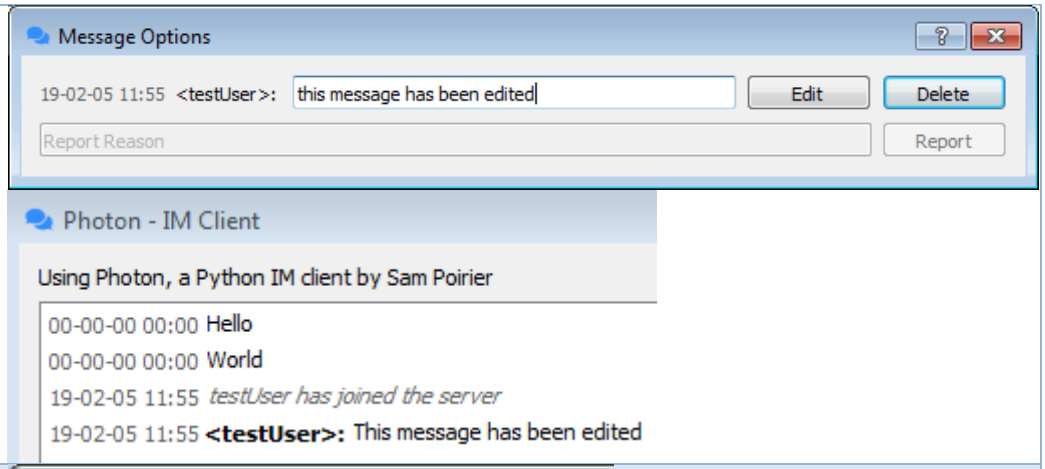
Test No.	11
Test Purpose	Can a large quantity of users connect at once?
Test Description	The server should be able to support a large quanitity of connected users at once
Test Data	a) Connecting 4 users simultaneously
Expected Result	a) 4 users connect with no issues
Actual Result	Pass – inline with expected results
Relavent Objectives	9

4.3 Test Evidence

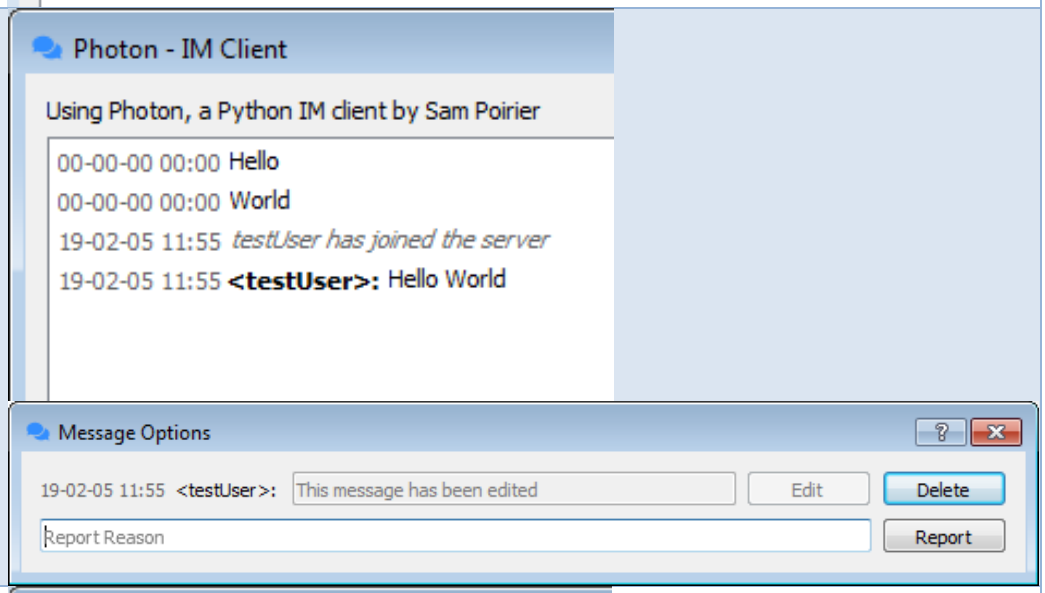
4.3.1 Screenshots

Test No.	Evidence
1a	
1b, 1c	
1d	
2a	
2b	

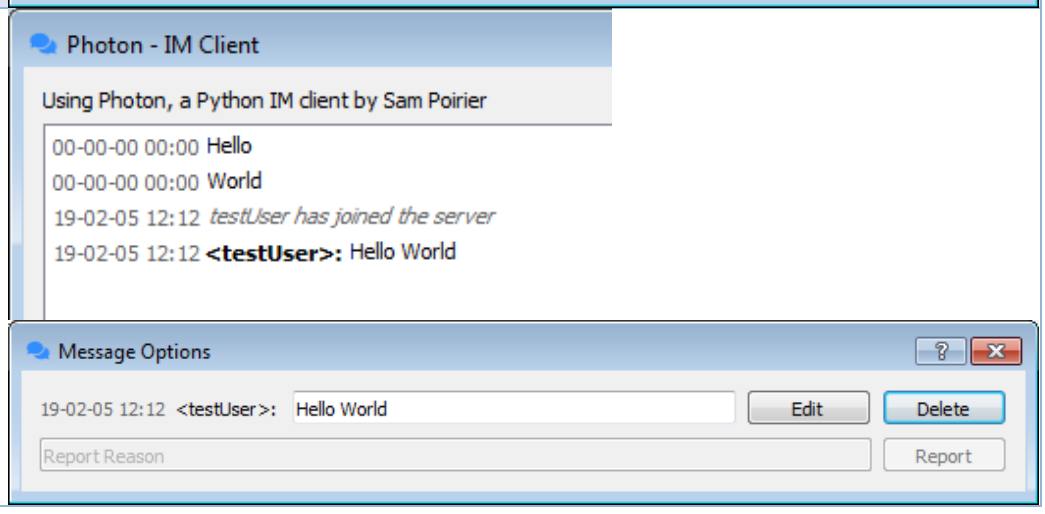
2c, 2d	
3a, 3b, 3c	
4	
5	
6a	

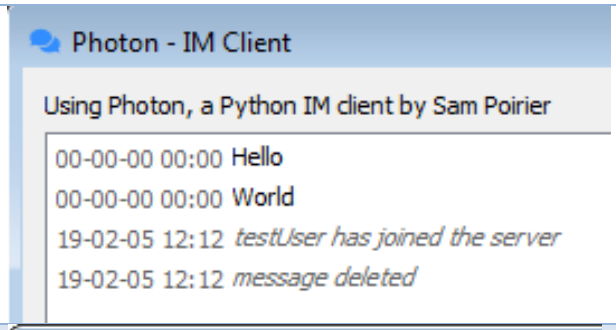

Message Options
19-02-05 11:55 <testUser>:
this message has been edited
Edit
Delete
Report Reason
Report
Photon - IM Client
Using Photon, a Python IM client by Sam Poirier
00-00-00 00:00 Hello
00-00-00 00:00 World
19-02-05 11:55 testUser has joined the server
19-02-05 11:55 <testUser>: This message has been edited

6b

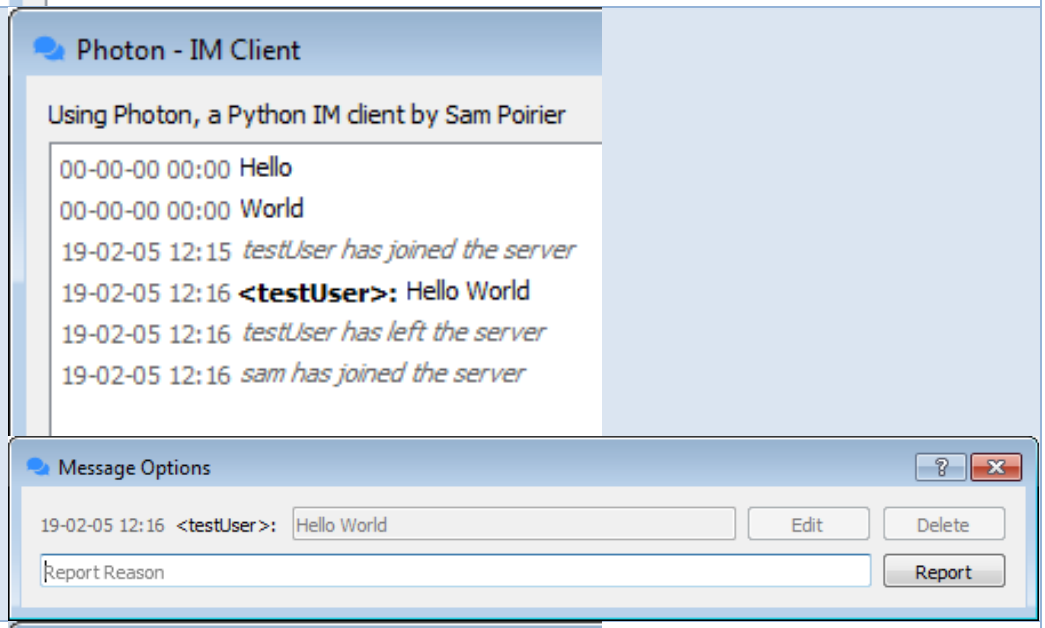

Photon - IM Client
Using Photon, a Python IM client by Sam Poirier
00-00-00 00:00 Hello
00-00-00 00:00 World
19-02-05 11:55 testUser has joined the server
19-02-05 11:55 <testUser>: Hello World
Message Options
19-02-05 11:55 <testUser>:
This message has been edited
Edit
Delete
Report Reason
Report

7a

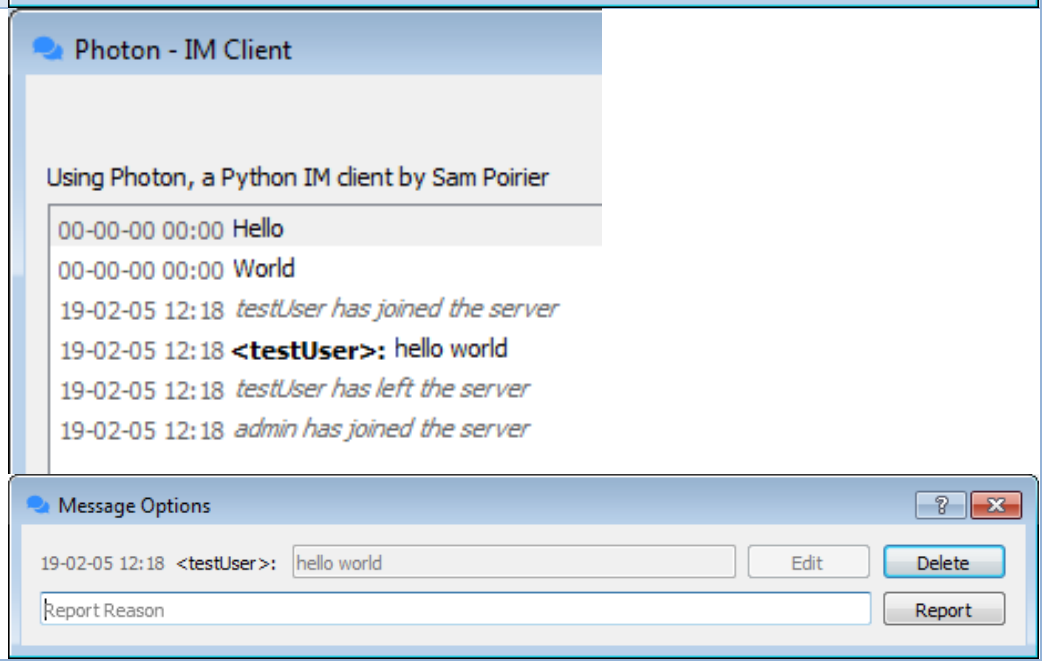

Photon - IM Client
Using Photon, a Python IM client by Sam Poirier
00-00-00 00:00 Hello
00-00-00 00:00 World
19-02-05 12:12 testUser has joined the server
19-02-05 12:12 <testUser>: Hello World
Message Options
19-02-05 12:12 <testUser>:
Hello World
Edit
Delete
Report Reason
Report

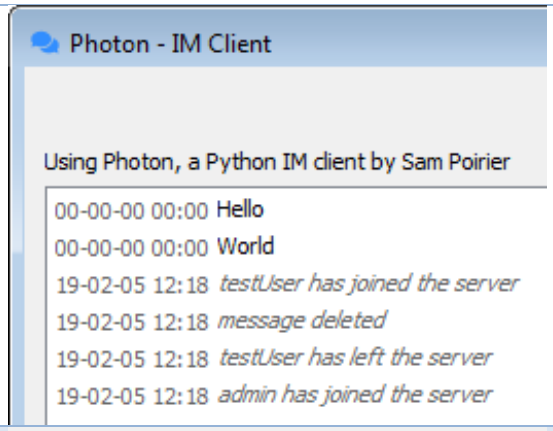

Photon - IM Client
Using Photon, a Python IM client by Sam Poirier
00-00-00 00:00 Hello
00-00-00 00:00 World
19-02-05 12:12 testUser has joined the server
19-02-05 12:12 message deleted

7b

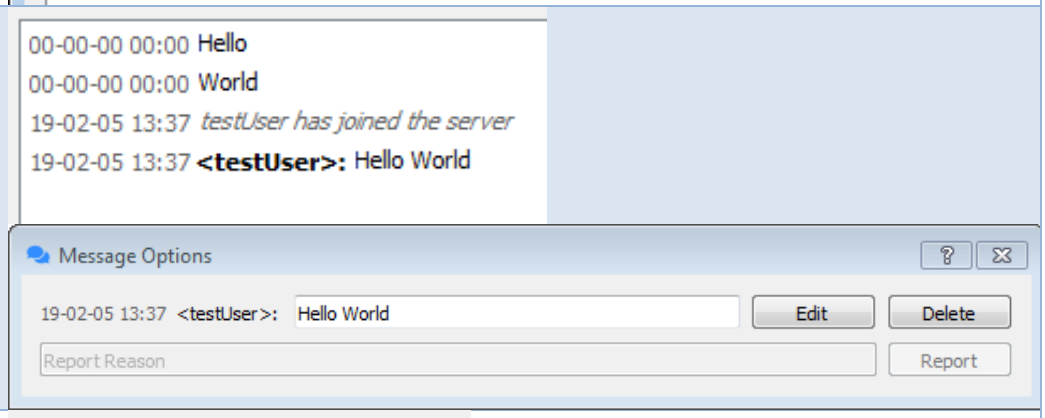

Photon - IM Client
Using Photon, a Python IM client by Sam Poirier
00-00-00 00:00 Hello
00-00-00 00:00 World
19-02-05 12:15 testUser has joined the server
19-02-05 12:16 <testUser>: Hello World
19-02-05 12:16 testUser has left the server
19-02-05 12:16 sam has joined the server
Message Options
19-02-05 12:16 <testUser>:
Hello World
Edit
Delete
Report Reason
Report

7c

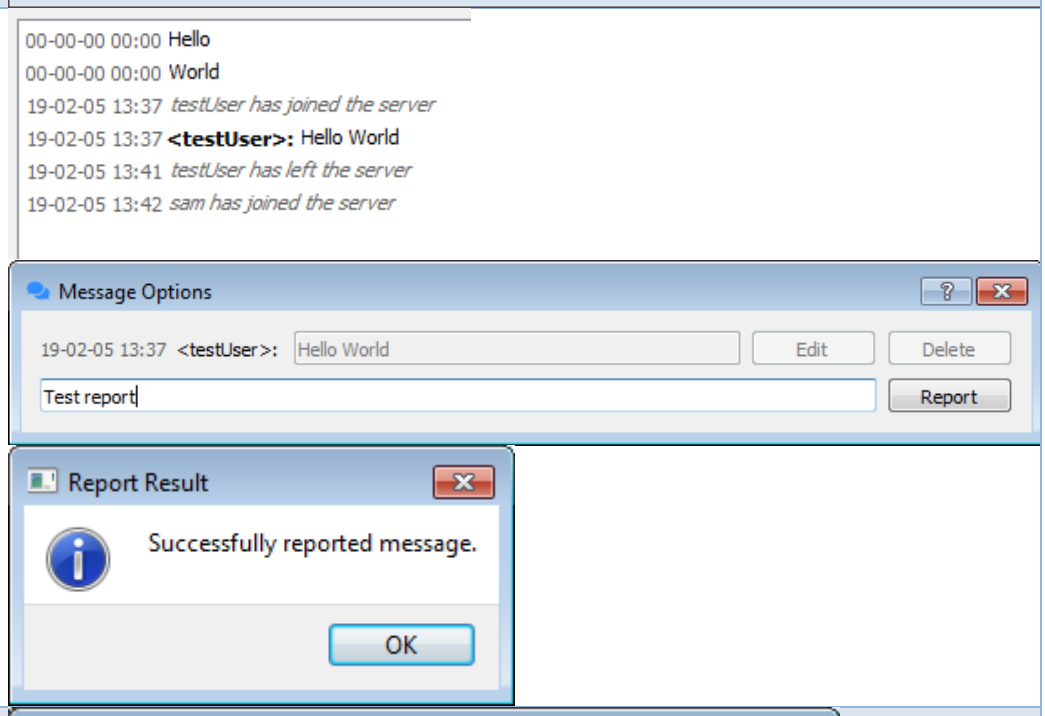

Photon - IM Client
Using Photon, a Python IM client by Sam Poirier
00-00-00 00:00 Hello
00-00-00 00:00 World
19-02-05 12:18 testUser has joined the server
19-02-05 12:18 <testUser>: hello world
19-02-05 12:18 testUser has left the server
19-02-05 12:18 admin has joined the server
Message Options
19-02-05 12:18 <testUser>:
hello world
Edit
Delete
Report Reason
Report


Photon - IM Client
Using Photon, a Python IM client by Sam Poirier
00-00-00 00:00 Hello
00-00-00 00:00 World
19-02-05 12:18 testUser has joined the server
19-02-05 12:18 message deleted
19-02-05 12:18 testUser has left the server
19-02-05 12:18 admin has joined the server

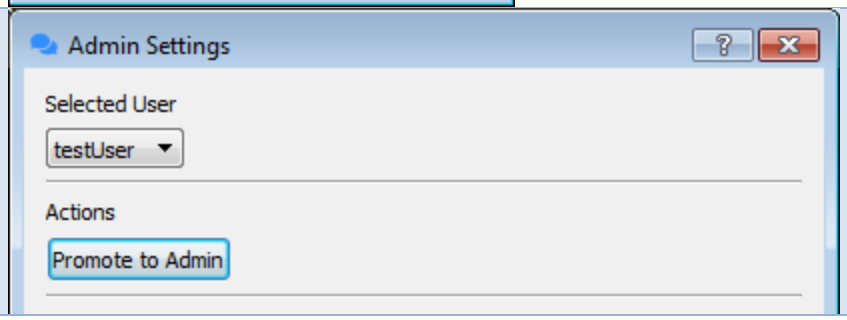
8a

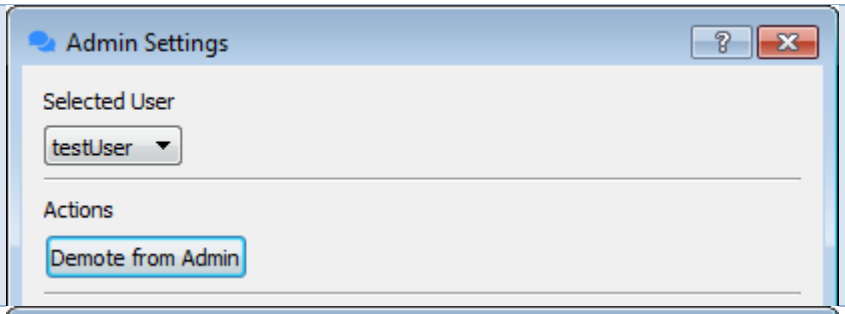

00-00-00 00:00 Hello
00-00-00 00:00 World
19-02-05 13:37 testUser has joined the server
19-02-05 13:37 <testUser>: Hello World
Message Options
19-02-05 13:37 <testUser>:
Hello World
Edit
Delete
Report Reason
Report

8b

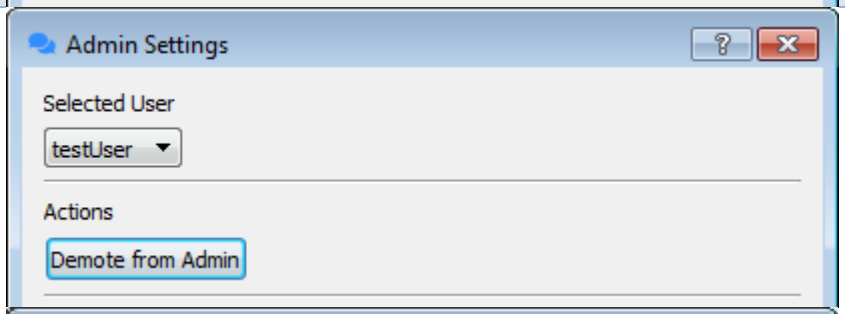

00-00-00 00:00 Hello
00-00-00 00:00 World
19-02-05 13:37 testUser has joined the server
19-02-05 13:37 <testUser>: Hello World
19-02-05 13:41 testUser has left the server
19-02-05 13:42 sam has joined the server
Message Options
19-02-05 13:37 <testUser>:
Hello World
Edit
Delete
Test report
Report
Report Result
Successfully reported message.
OK

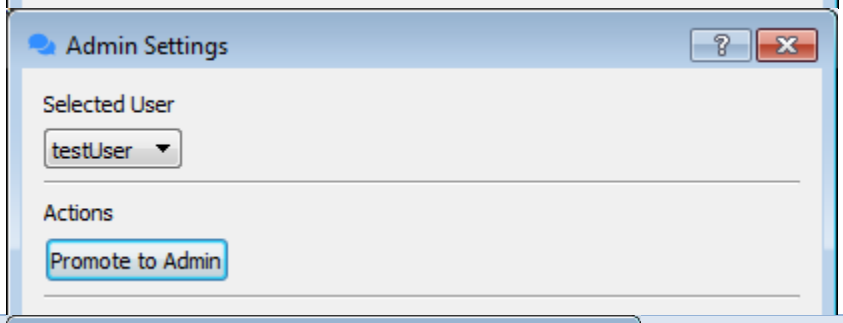
9a


Admin Settings
Selected User
testUser
Actions
Promote to Admin

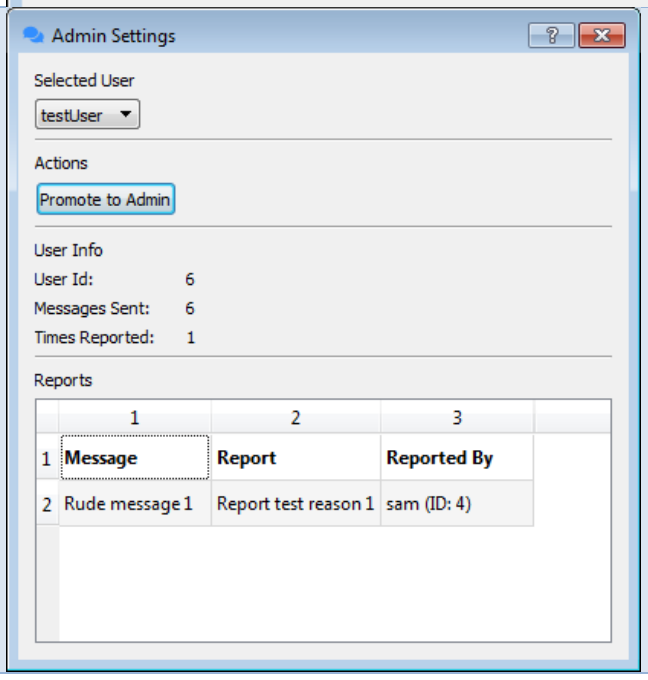

Admin Settings
Selected User
testUser
Actions
Demote from Admin

9b

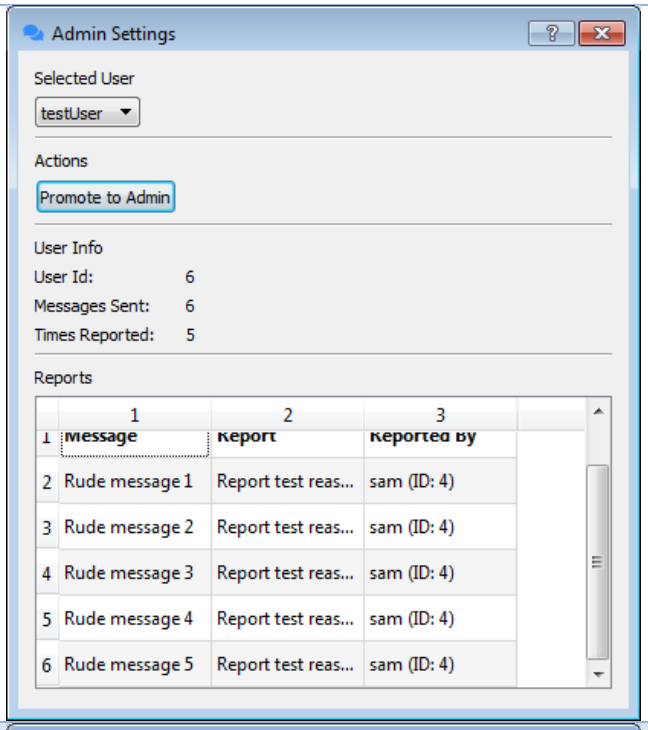

Admin Settings
Selected User
testUser
Actions
Demote from Admin


Admin Settings
Selected User
testUser
Actions
Promote to Admin

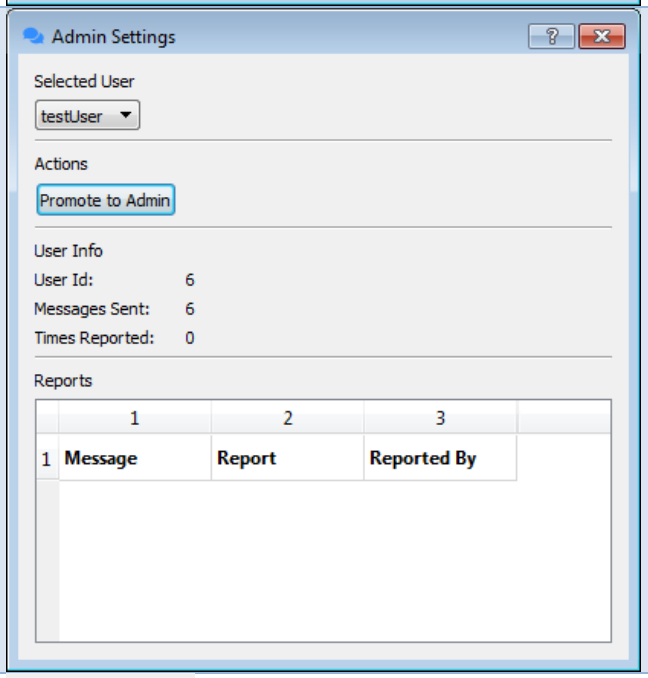
10a


Admin Settings
Selected User
testUser
Actions
Promote to Admin
User Info
User Id: 6
Messages Sent: 6
Times Reported: 1
Reports
1 2 3
1 Message Report Reported By
2 Rude message 1 Report test reason 1 sam (ID: 4)

10b


Admin Settings
Selected User
testUser
Actions
Promote to Admin
User Info
User Id: 6
Messages Sent: 6
Times Reported: 5
Reports
1 2 3
1 Message Report Reported By
2 Rude message 1 Report test reas... sam (ID: 4)
3 Rude message 2 Report test reas... sam (ID: 4)
4 Rude message 3 Report test reas... sam (ID: 4)
5 Rude message 4 Report test reas... sam (ID: 4)
6 Rude message 5 Report test reas... sam (ID: 4)

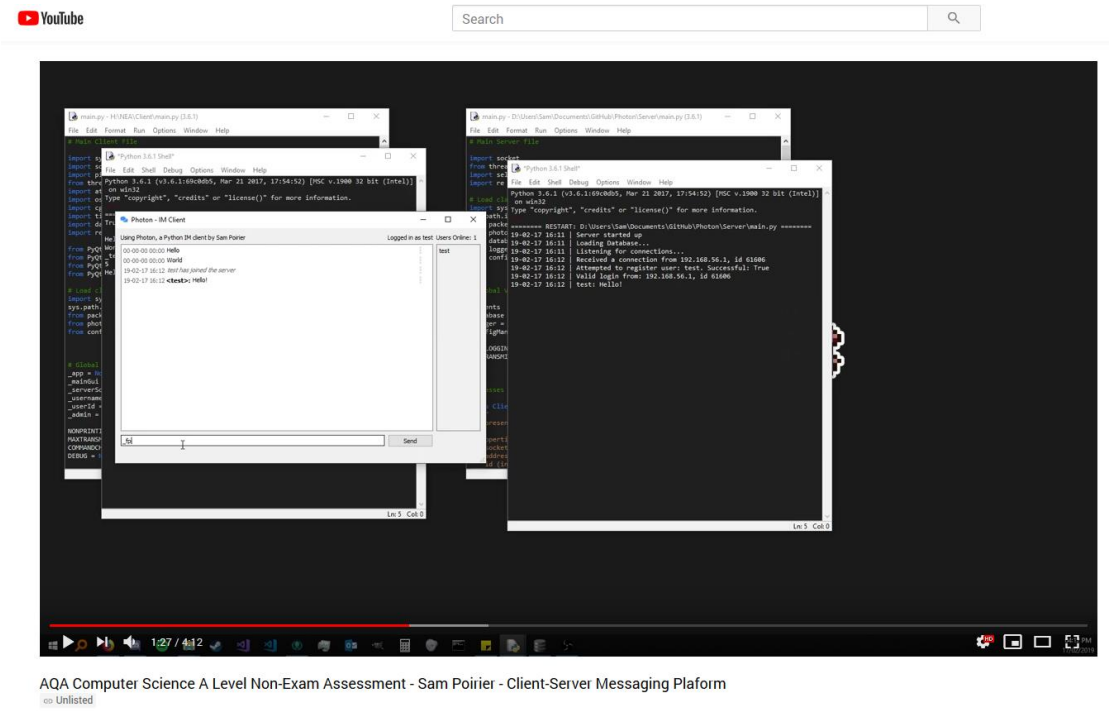
10c


Admin Settings
Selected User
testUser
Actions
Promote to Admin
User Info
User Id: 6
Messages Sent: 6
Times Reported: 0
Reports
1 2 3
1 Message Report Reported By

11


Users Online: 4
user1
user2
user3
user4

4.3.2 Video Evidence



A video showcasing the project in operation can be found on YouTube:

https://www.youtube.com/watch?v=eeGppVMGFhU

4.4 End-user Testing

Program was given to a tester who ran it and tried to break it for around 20 minutes. They wrote their attempts to break the program as mini test tables.

User Test No.	1
Test Purpose	Can I login?
Test Description	Attempting to log in to an account.
Feedback	Working very well; no complaints. Very intituative and fast to user. No crashes or bugs discovered.
Result	Pass

User Test No.	2
Test Purpose	Can I login to an account if there are two users with the same name?
Test Description	Created two accounts with the same username; can I login?
Feedback	No, the program crashed
Result	Fail

User Test No.	3
Test Purpose	Can I connect multiple users?
Test Description	Created two accounts with the same username; can I login?

Feedback	Multiple users have worked perfectly fine.
Result	Pass

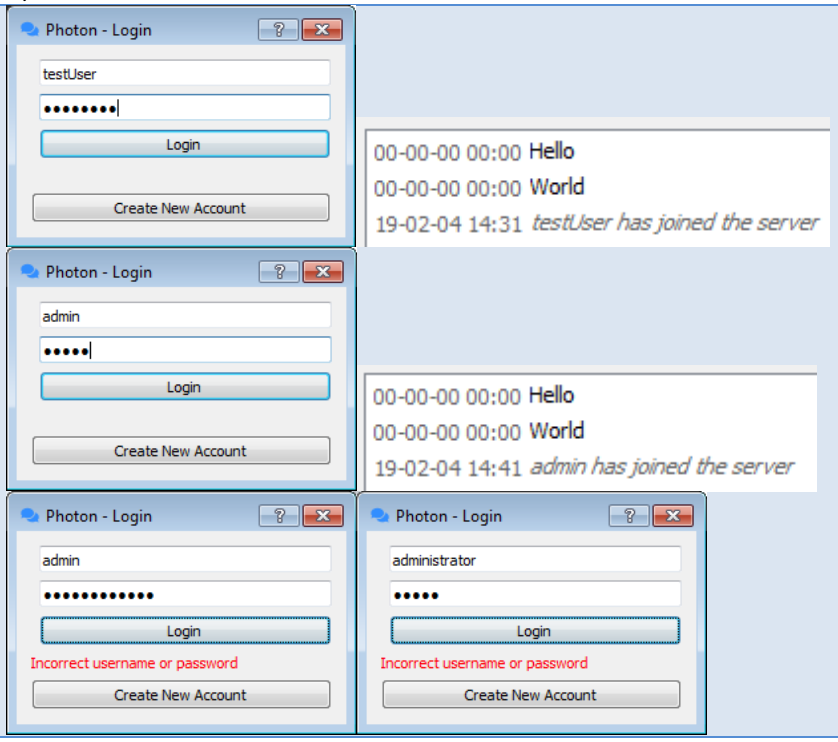
User Test No.	4
Test Purpose	Can I report messages?
Test Description	How well does reporting others and my own messages work?
Feedback	Reporting works fine (maybe grey is slightly hard to see for the button?).
Result	Pass

User Test No.	5
Test Purpose	Can I edit messages?
Test Description	How well does editing others and my own messages work?
Feedback	Editing messages works well. Can't edit messages that aren't mine - this is good.
Result	Pass

User Test No.	6
Test Purpose	Can I delete messages?
Test Description	How well does deleting others and my own messages work?
Feedback	Deleting messages works well. Can't delete messages that aren't mine - this is good.
Result	Pass

4.5 Failed Test/User Feedback Improvements

Test No.	2
Test Purpose	Can users login to accounts?
Test Description	When the program is started, determine if the user can successfully login with their account.
Test Data	a) Username: “testUser”, Password: “testPassword” b) Username: “admin”, Password: “admin” c) Username: “admin”, Password: “testPassword” d) Username: “administrator”, Password: “admin”
Expected Result	a) Successful user login b) Successful user login c) Unsuccessful login, incorrect password error appears d) Unsuccessful login, unknown user error appears
Actual Result	a) Pass b) Pass c) Fail – no error message was displayed d) Fail – no error message was displayed
Relavent Objectives	1a, 1b, 1c, 1d
Fail Details	For some reason, the unsucuccesful login error is not being displayed in the the client login window. There are a number of different places that the error could lie, including the server not sending it correctly, the client

	not receiving it properly, and the client not displaying it properly. All of which must be investigated.
Actual Cause	Server response packet was not being constructed properly; the error message parameter was being assigned to the wrong property.
Retest Results	a) Pass b) Pass c) Pass d) Pass
Retest Evidence a, b, c, d	

5 Evaluation

5.1 Objectives Analysis

No	Objective	Performance Criteria	Evaluation
1	Login Screen User Interface must contain: e) Input for username and password f) Login Button g) Open Register Window Button h) Error Label to display login issues	All of these UI elements must exist in an organised and easy to read layout.	*Completley Acheived* The client user interface contains all the specified elements which work successfully in the intended manner. End users had no difficulties navigating the menus.
2	Register User Screen UI must contain: d) Input for username,	All of these UI elements must exist in an organised and easy	*Completley Acheived* The user interface contains all the elements specified,

	password and password confirmation e) Register (submit) button f) Error label to display register issues	to read layout.	and works successfully in the intended manner. End users had no difficulties navigating the menus.
3	Main Window UI must contain: e) Box to contain message history f) Input box for sending messages g) Button to send message h) General server information; who's connected, who you're logged in as etc	All of these UI elements must exist in an organised and easy to read layout.	*Completley Acheived* End users navigated the UI with ease, and all elements functioned in the intended manner and were included correctly.
4	Connecting to the server should be secure.	The initial handshake between client and server should be secure; ie no 'fake' clients attempting to connect should be accepted.	*Completley Acheived* Only clients that perform precisely the correct operations are accepted by the server; any deviation and the connection will be dropped.
5	Client/Server Communication should be secure.	All packets to and from the server should be secure so that anyone intercepting them cannot read the information.	*Completley Acheived* All packets are pickeled securely before being sent between the client and server.
7	User interface should be easy to use and read.	The UI should be intuitive to read, use and understand.	*Completley Acheived* No users reported issues understanding the UI.
8	Client and server should be easy to set up and run.	The client and server should require minimal setup to get into full working/running order.	*Completley Acheived* Minimal setup is needed, simply configure config.json if settings other than default are wanted, and run the two scripts.
9	The server must be able to handle lots of users both consecutively and in general.	The server must be able to support multiple concurrently connected users and a larger amount of total registered users.	*Completley Acheived* The server can handle multiple connected users and an even large amount of created user accounts.
10	There must be a way for users to configure settings for the client.	The clients should be configurable via a settings menu which is	*Partially Acheived* The settings are not extensive enough and are

		saved between sessions.	quite technical, meaning a menu system is unneccesary.
11	The application must support most requested features by the client.	As many features requested by the client as possible should be implemented into the program.	*Completley Acheived* All feasible client requests were implemented.
12	Users should be able to edit messages	All messages sent by the user should be editable by them, but users should not be able to edit other's messages.	*Completley Acheived* Users can successfully edit their own messages, but not those of others.
13	Users should be able to report messages	Users should be able to report messages in case they contain rude/innapropriate words etc.	*Completley Acheived* Users can report other's messages, but not their own.
14	Admins should be able to see user reports	The admin should have an administration menu which they can use to review the reports against users.	*Completley Acheived* Admins can successfully view reports against a specific user via admin settings menu.
15	Admins should be able to see user statistics	Admin accounts should be able to see basic statistics for each user account.	*Completley Acheived* Admins can successfully view user stats via admin settings menu.
16	Admins should be able to promote/demote users	Admin accounts should be able to give admin permissions to other accounts and remove admin permissions from other accounts.	*Completley Acheived* Admins can promote or demote users successfully. Note: Requires the user to relog to take effect due to structure of connection logic.
17	Admins should be able to delete all messages	Admin accounts should be able to delete all messages as they could contain rude/explicit content.	*Completley Acheived* Admins can delete all user messages successfully.
18	Logged in users list should be displayed alphabetically	The list of online users should always be automatically sorted to display connected users in alphabetic order rather than join order.	*Completley Acheived* Admins can delete all user messages successfully.

19	Users should be able to style their messages	Users should be able to format their messages to feature bold, italics etc using a markdown like format (surrounding it with symbols, eg _italics_ would display as *italics*).	*Completley Acheived* All messages support styling via markdown-esque tags.
20	Hovering over messages should highlight them	Hovering over a message should highlight the background making it easier to read and determine which one you are interacting with.	*Completley Acheived* When a message is rolled-over, it highlights successfully.

5.2 End User Feedback

User Comment	Actions Taken
Login system: working very well no complaints. Very intituative and fast to user.	*Positive feedback, none needed*
Chat message: quite annoying that program does not autoscroll down when users adds a message	Updated program to autoscroll window when a new message arrives.
Messages are saved from before which I like.	*Positive feedback, none needed*
Multiple users have worked perfectly fine.	*Positive feedback, none needed*
Highlighting messages is a nice touch.	*Positive feedback, none needed*
Can only select message contents which is really nice.	*Positive feedback, none needed*
Editing messages works well.	*Positive feedback, none needed*
Deleting messages works well.	*Positive feedback, none needed*
No indicaion for editted messages (is there supposed to be one?)	Added an ‘(edited)’ marker to messages when they are edited.
Can't edit or delete messages that aren't mine this is good.	*Positive feedback, none needed*